

Arrays are indexed sets.

Pairs and Maps

- Let A and B be sets. The Cartesian product of A and B , denoted by $A \times B$, is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$:

$$A \times B = \{ (a,b) \mid a \in A \text{ and } b \in B \}$$

- A map is an associative container, whose elements are key-value pairs. The key serves as an index into the map, and the value represents the data being stored and retrieved.

Associative Array (Dictionaries)

- An associate array is a map in which elements are indexed by a key rather than by their position.

$$a[i] = \begin{cases} v, & \text{if } i \mapsto v \text{ in } a \\ \perp, & \text{otherwise} \end{cases}$$

- Example:

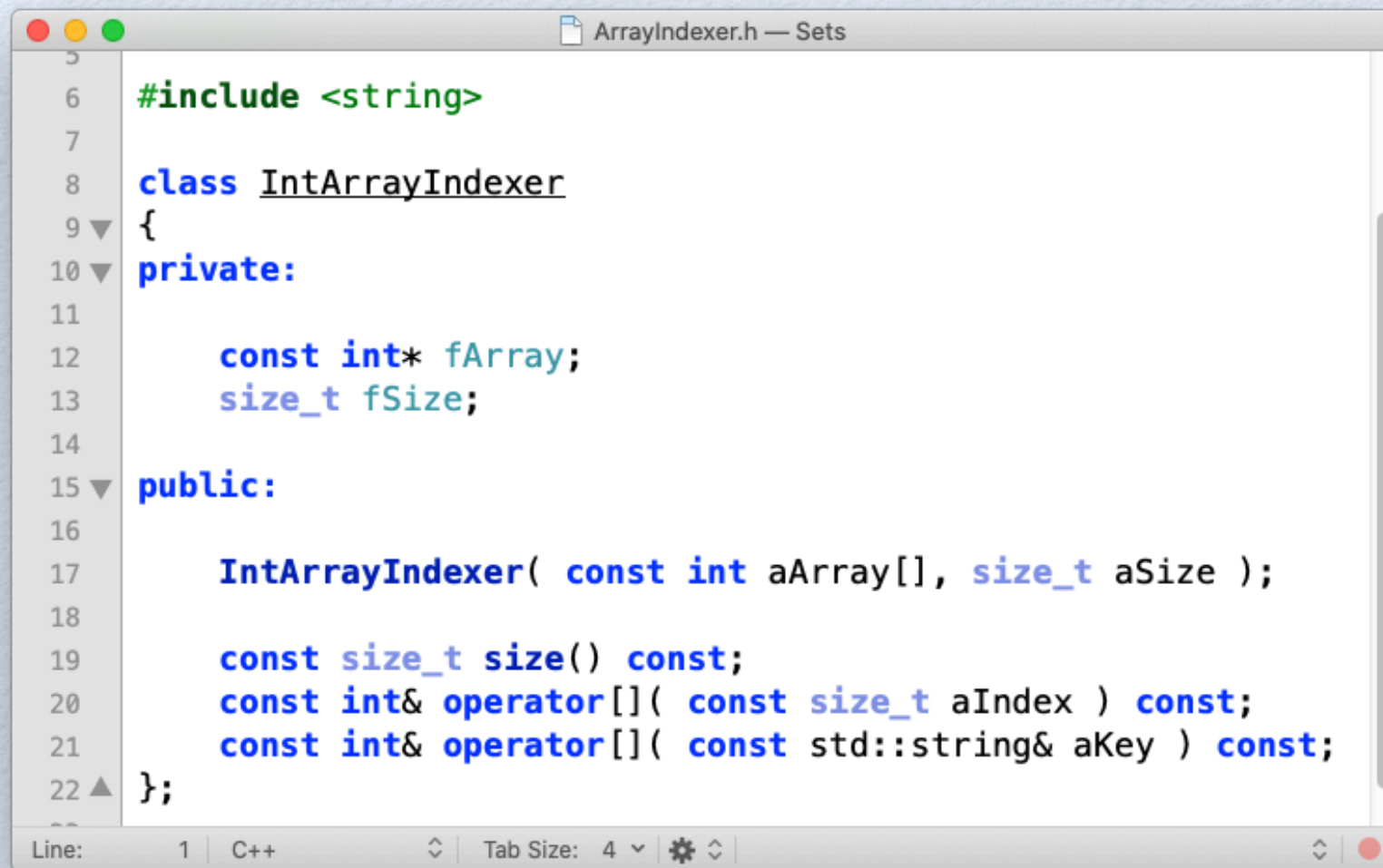
$$a = \{ ("u" \mapsto 345), ("v" \mapsto 2), ("w" \mapsto 39), ("x" \mapsto 5) \}$$

$$a["w"] = 39$$

$$a["z"] = \perp$$

From Indices to Keys

- We can define an adapter class that defines an indexer:



```
5
6 #include <string>
7
8 class IntArrayIndexer
9 {
10 private:
11
12     const int* fArray;
13     size_t fSize;
14
15 public:
16
17     IntArrayIndexer( const int aArray[], size_t aSize );
18
19     const size_t size() const;
20     const int& operator[]( const size_t aIndex ) const;
21     const int& operator[]( const std::string& aKey ) const;
22 };
23
```

The screenshot shows a code editor window titled "ArrayIndexer.h — Sets". The code defines a class `IntArrayIndexer` with private attributes `fArray` and `fSize`, and public methods for construction, size retrieval, and indexing by both index and key. The editor interface includes a line number margin on the left and a status bar at the bottom showing "Line: 1 C++" and "Tab Size: 4".

Indexer Constructor

Arrays are passed as pointers to the first element to functions in C++.

```
3
4  #include "ArrayIndexer.h"
5
6  #include <stdexcept>
7
8  using namespace std;
9
10 IntArrayIndexer::IntArrayIndexer( const int aArray[], size_t aSize ) :
11     fArray(aArray),
12     fSize(aSize)
13 {}
14
```

Line: 1 C++ Tab Size: 4

We must use member initializer to initialize const instance variables!

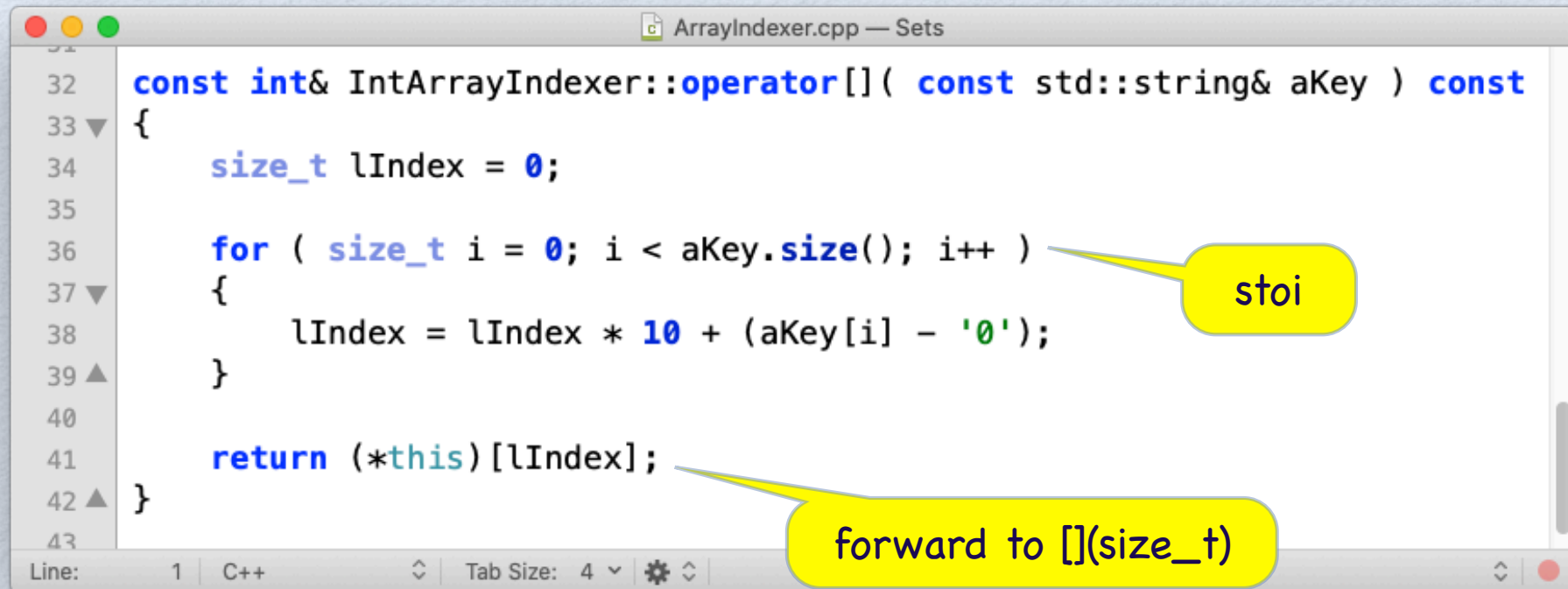
Basic Indexer Operations

```
ArrayIndexer.cpp — Sets
15  const size_t IntArrayIndexer::size() const
16  {
17      return fSize;
18  }
19
20  const int& IntArrayIndexer::operator[]( const size_t aIndex ) const
21  {
22      if ( aIndex < fSize )
23      {
24          return fArray[aIndex];
25      }
26      else
27      {
28          throw out_of_range( "Illegal array index." );
29      }
30  }
31
```

One-sized range test.

Line: 1 C++ Tab Size: 4

The Indexer



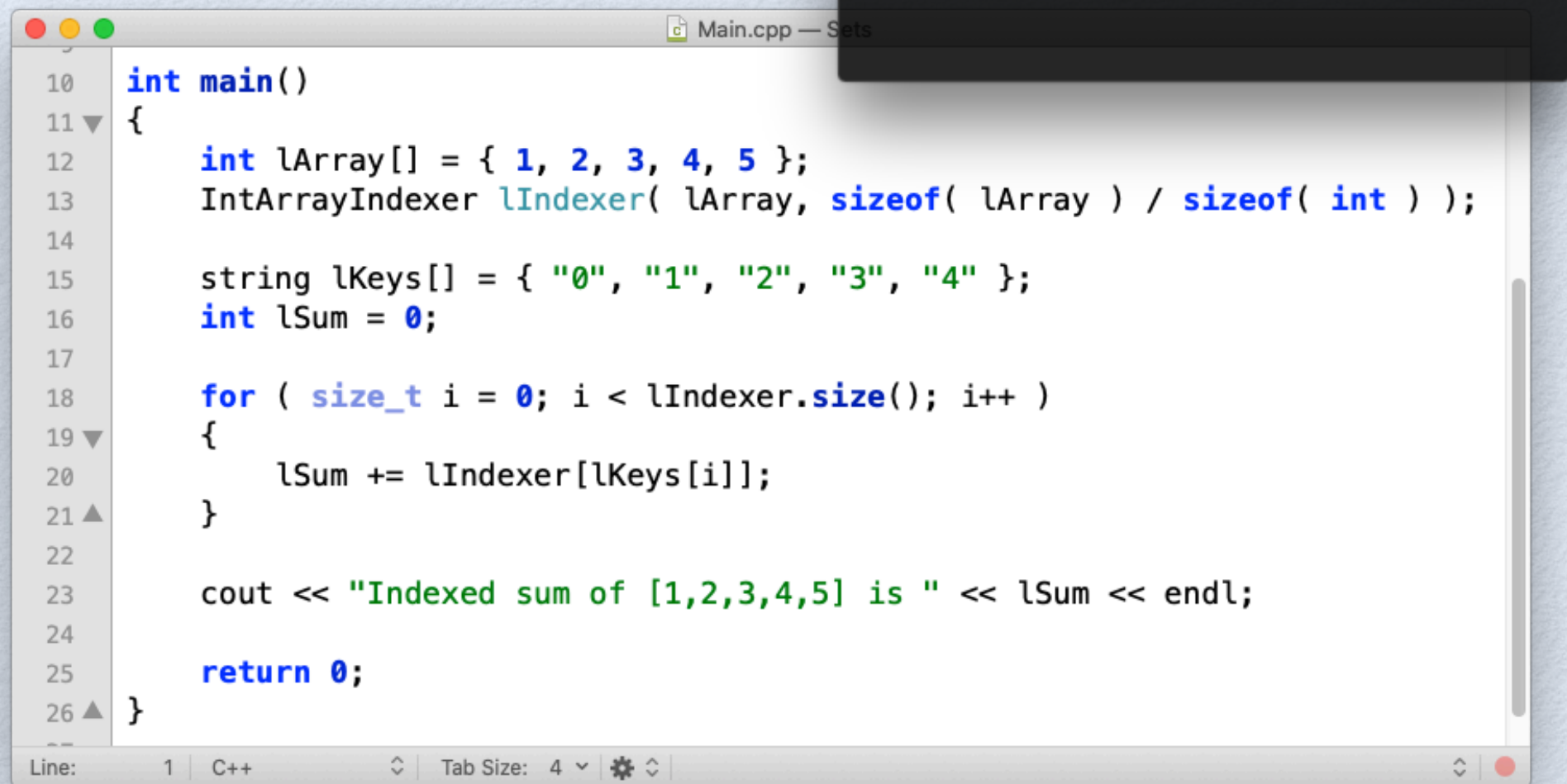
```
32 const int& IntArrayIndexer::operator[]( const std::string& aKey ) const
33 {
34     size_t lIndex = 0;
35
36     for ( size_t i = 0; i < aKey.size(); i++ )
37     {
38         lIndex = lIndex * 10 + (aKey[i] - '0');
39     }
40
41     return (*this)[lIndex];
42 }
```

stoi

forward to [](size_t)

- We use the **const specifier** to indicate that the operator[]:
 - is a read-only getter
 - does not alter the elements of the underlying collection
- We use a **const reference** to avoid copying the original value stored in the underlying collection.

Testing the Indexer



```
10 int main()
11 {
12     int lArray[] = { 1, 2, 3, 4, 5 };
13     IntArrayIndexer lIndexer( lArray, sizeof( lArray ) / sizeof( int ) );
14
15     string lKeys[] = { "0", "1", "2", "3", "4" };
16     int lSum = 0;
17
18     for ( size_t i = 0; i < lIndexer.size(); i++ )
19     {
20         lSum += lIndexer[lKeys[i]];
21     }
22
23     cout << "Indexed sum of [1,2,3,4,5] is " << lSum << endl;
24
25     return 0;
26 }
```

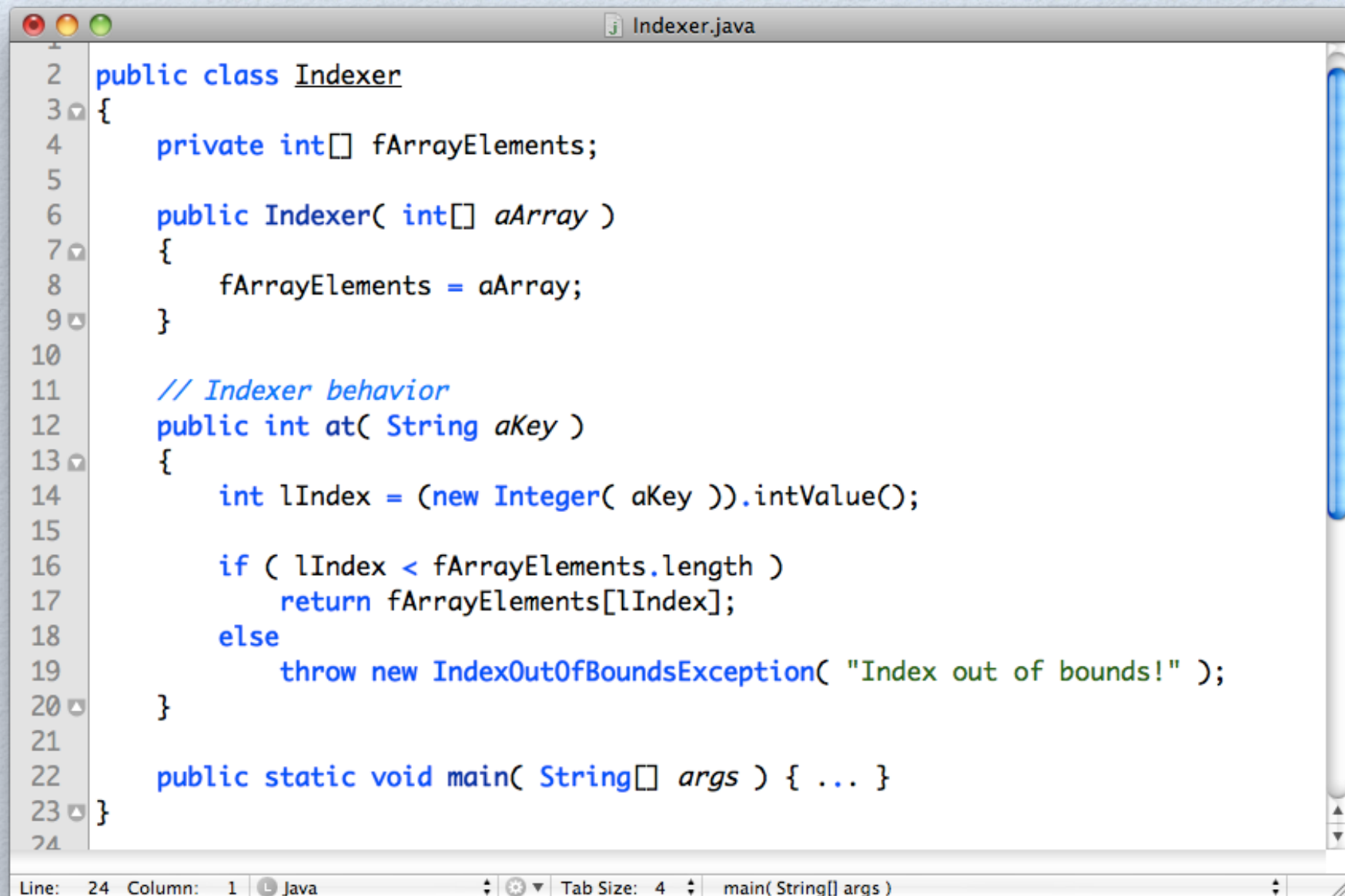
Kamala: COS3008 Markus\$./ArrayIndexer
Indexed sum of [1,2,3,4,5] is 15
Kamala: COS3008 Markus\$ _

**How can we define an
indexer in Java?**

The Transition to Java

- We need to define an Indexer class.
- Java does not support operator overloading. So, we need to map [] to a member function.
- The built-in type `Integer` provides the required conversion operations.
- We use `IndexOutOfBoundsException` to signal an index error.

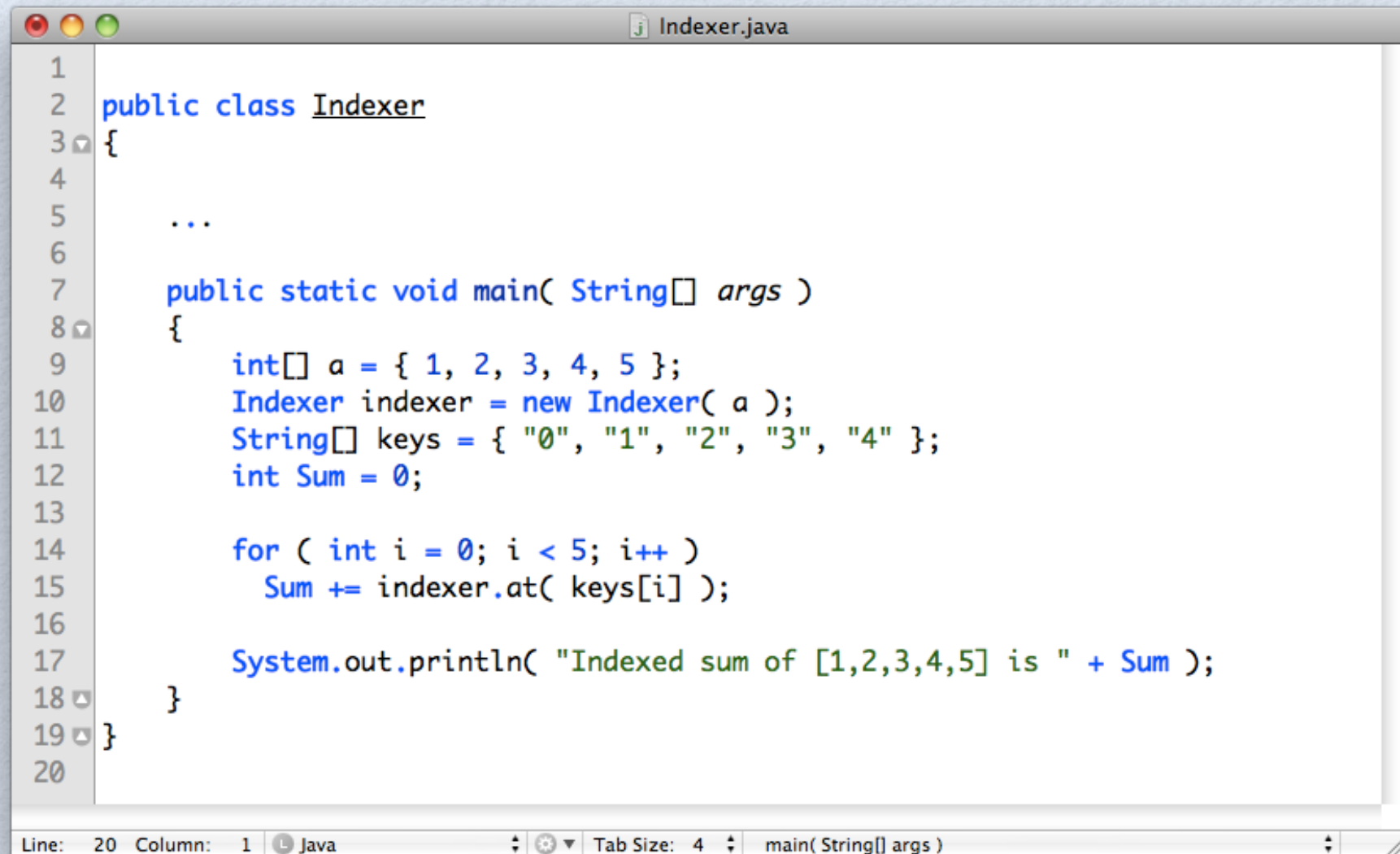
Indexer's at(String aKey) Method



```
1
2 public class Indexer
3 {
4     private int[] fArrayElements;
5
6     public Indexer( int[] aArray )
7     {
8         fArrayElements = aArray;
9     }
10
11     // Indexer behavior
12     public int at( String aKey )
13     {
14         int lIndex = (new Integer( aKey )).intValue();
15
16         if ( lIndex < fArrayElements.length )
17             return fArrayElements[lIndex];
18         else
19             throw new IndexOutOfBoundsException( "Index out of bounds!" );
20     }
21
22     public static void main( String[] args ) { ... }
23 }
24
```

Line: 24 Column: 1 Java Tab Size: 4 main(String[] args)

The Indexer's main Method



```
1
2 public class Indexer
3 {
4
5     ...
6
7     public static void main( String[] args )
8     {
9         int[] a = { 1, 2, 3, 4, 5 };
10        Indexer indexer = new Indexer( a );
11        String[] keys = { "0", "1", "2", "3", "4" };
12        int Sum = 0;
13
14        for ( int i = 0; i < 5; i++ )
15            Sum += indexer.at( keys[i] );
16
17        System.out.println( "Indexed sum of [1,2,3,4,5] is " + Sum );
18    }
19 }
20
```

Line: 20 Column: 1 Java Tab Size: 4 main(String[] args)