Iterator Design Pattern

CSCI 2300

Consider a Linked List data structure

```
public class MyLinkedList<T>
{
   private Node<T> head;
   private Node<T> current;
   public T getFirst(){…}
   public T getNext(){…}
   public void insert(T element){…}
   public int getLength(){…}

   private class Node<T>
   {
      private T element;
      private T next;
      public T getValue(){…}
      public T getNext(){…}
   }
}
```

Example of using MyLinkedList

```
MyLinkedList<String> myList;
myList.insert("Joe");
myList.insert("Bob");
myList.insert("Sam");
String first = myList.getFirst();
String second = myList.getNext();
String third = myList.getNext();
```

What best describes the operating of iterating over MyLinkedList?

A. Iterating through MyLinkedList should be a read-only operation; instead it changes the internal state of the object

B. Iterating through MyLinkedList is a read-only operation

C. Iterating through MyLinkedList should not be a read-only operation: it should modify the internal state of the object

D. Iterating through MyLinkedList changes the internal state of the object; this should be fixed by returning changing `getFirst()` and `getNext()` to return `Node<T>` instead of `T`

E. None of the above

Key Elements of Iterator Pattern

- A class that represents a collection of objects
- Iterator interface
  - Java's Iterator interface
  - Your own Iterator interface
- When implementing an Iterator Pattern include:
  - A way to create an iterator of a collection
  - hasNext() method – true if there is another element in the collection
  - next() method – retrieves the next element of the collection
  - Optionally: remove() method – to remove the current element of the collection

Which of the following classes would benefit from implementing Iterator Pattern?

A. BankAccount
B. TicTacToePlayer
C. List – (list of students in a class, for example)
D. BookShelf
E. All of the above

Java's Iterator pattern interfaces

- Enumeration<E>
  - Access to each element of the collection
- Iterator<E>
  - Access to each element of the collection
  - Access to remove elements from the collection

java.util.Enumeration<E>

Modifier and Type	Method and Description
`boolean`	**`hasMoreElements()`** Tests if this enumeration (collection) contains more elements
`E`	**`nextElement()`** Returns the next element of this enumeration if this enumeration has at least one more element to provide. Throws NoSuchElementException – if no more elements exist

Suppose we have a class, `History`, containing a list of `Record` objects. This class implements `java.util.Enumeration<Record>` interface. What does **`nextElement()`** method of this class return?

A. History

B. Enumeration

C. Record

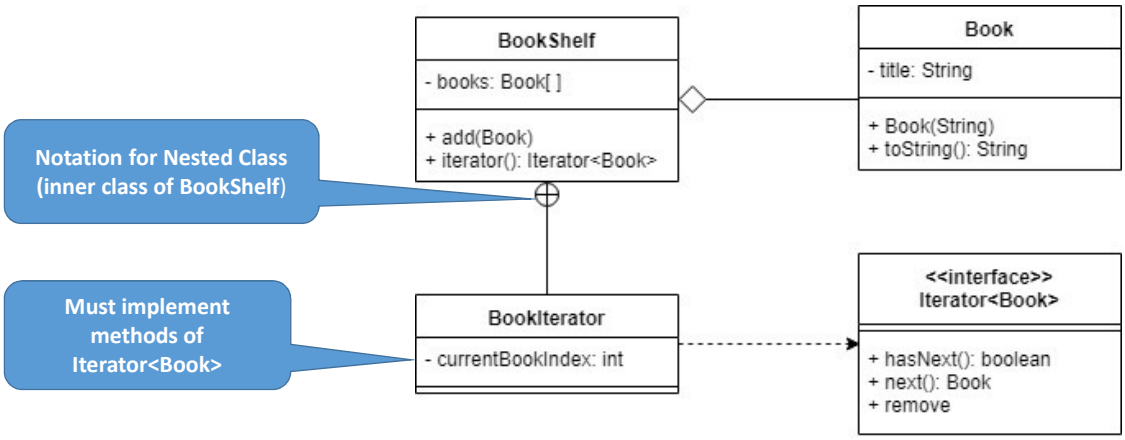
D. Boolean

E. None of the above

java.util.Iterator<E>

Modifier and Type	Method and Description
`boolean`	**`hasNext()`** Returns true if the iteration has more elements
`E`	**`next()`** Returns the next element of this iteration. Throws NoSuchElementException – if no more elements exist
`void`	**`remove()`** Removes from the collection the last element returned by the iterator (optional)

Iterator Pattern Design



Iterator Pattern Implementation

```
public class BookShelf
{
   // constructor
   public BookShelf(){…}
   public void add(Book b){…}
   public Iterator<Book> iterator(){
      return new BookIterator();
  }
  // BookIterator class here (from next slide)
}
```

BookIterator implementation

```
public class BookIterator implements Iterator<Book>
{
    // instance variables
    // current book index for example

    // constructor
    public BookIterator(){…}

    public boolean hasNext(){…}
    public Book next(){…}
}
```

Iterating over a BookShelf

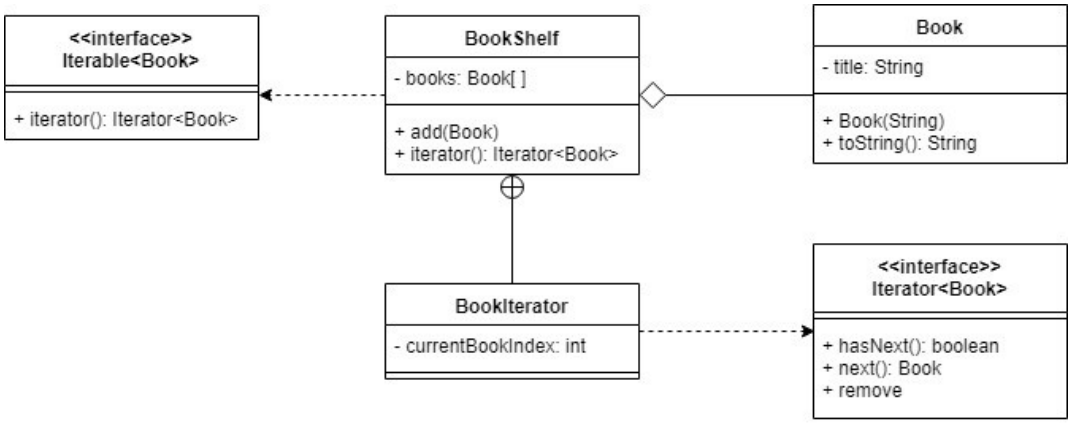
```
BookShelf shelf = new BookShelf();
shelf.add(new Book("Object Oriented Design"));
shelf.add(new Book("The Adventures of Tom Sawyer"));

Iterator<Book> bookIter = shelf.iterator();
while (bookIter.hasNext())
{
   System.out.println(bookIter.next());
}
```

`Iterable` interface

- In our example, we can iterate over a BookShelf
- The BookShelf is "iterable"
- Utilize Iterable interface, to improve design

Iterable BookShelf Design



Iterable BookShelf Implementation

```
public class BookShelf implements Iterable<Book>
{
   public BookShelf(){…}
   public void add(Book b){…}
   public Iterator<Book> iterator(){
      return new BookIterator();
  }
  // BookIterator class here
}
```

Iterating over an 'Iterable' BookShelf

```
BookShelf shelf = new BookShelf();
shelf.add(new Book("Object Oriented Design"));
shelf.add(new Book("The Adventures of Tom Sawyer"));

for(Book b: shelf)
{
   System.out.println(b);
}
```

Lab 13

- Search Java documentation for a class that implements an Iterable interface
- Write a Driver.java that uses the class you found to iterate over a collection of objects
  - Use the shorthand notation, as in the BookShelf example:
    `for(Book b: shelf)`