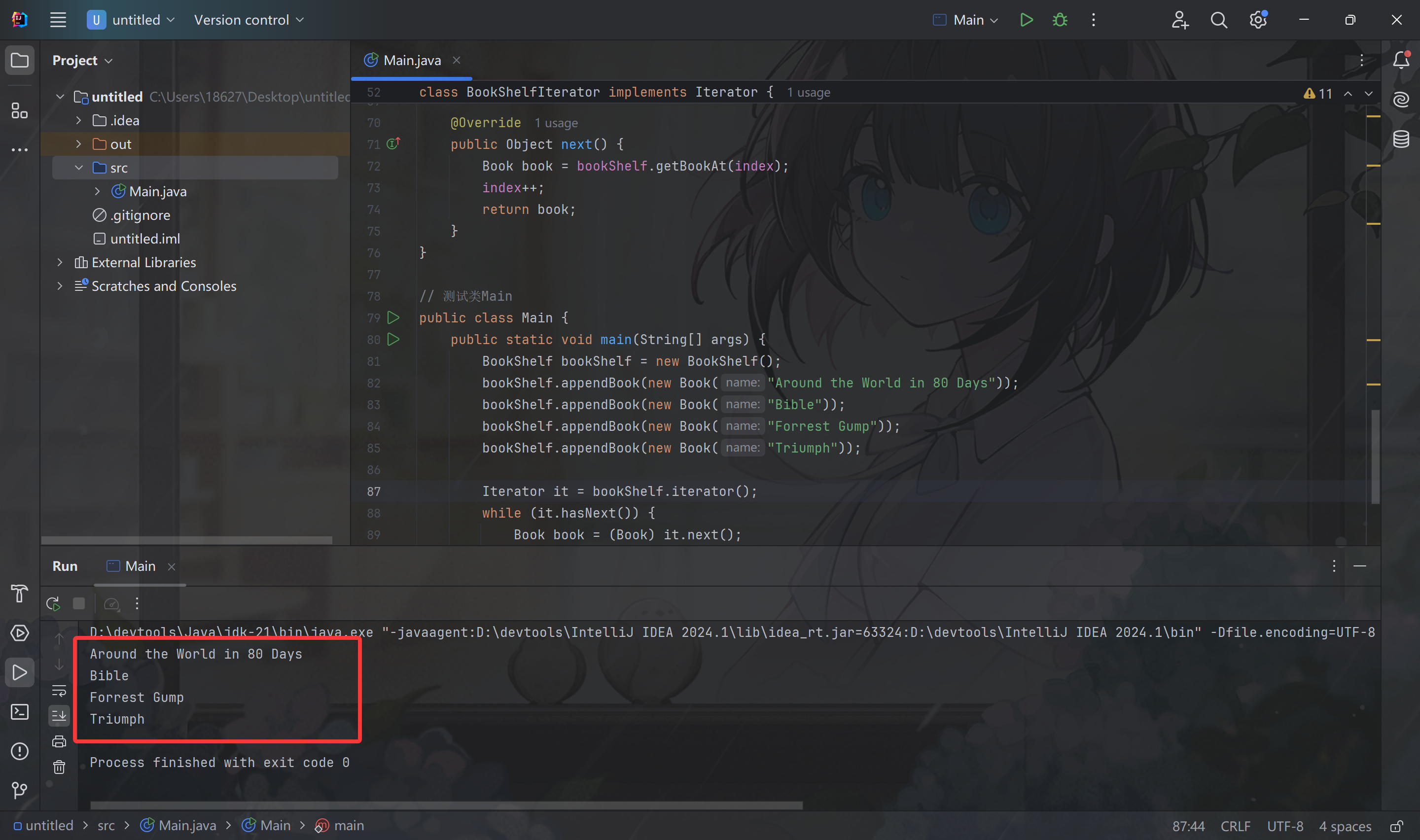
1. 针对Iterator的例子，将存储Book用的数组换成其他Collection并运行

Code：

// 定义Aggregate接口  
interface Aggregate {  
 public abstract Iterator iterator();  
}  
  
// 定义Iterator接口  
interface Iterator {  
 public abstract boolean hasNext();  
 public abstract Object next();  
}  
  
// Book类  
class Book {  
 private String name;  
  
 public Book(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
}  
  
// BookShelf类，使用ArrayList存储Book  
class BookShelf implements Aggregate {  
 private java.util.ArrayList<Book> books;  
  
 public BookShelf() {  
 this.books = new java.util.ArrayList<>();  
 }  
  
 public Book getBookAt(int index) {  
 return books.get(index);  
 }  
  
 public void appendBook(Book book) {  
 books.add(book);  
 }  
  
 public int getLength() {  
 return books.size();  
 }  
  
 @Override  
 public Iterator iterator() {  
 return new BookShelfIterator(this);  
 }  
}  
  
// BookShelfIterator类  
class BookShelfIterator implements Iterator {  
 private BookShelf bookShelf;  
 private int index;  
  
 public BookShelfIterator(BookShelf bookShelf) {  
 this.bookShelf = bookShelf;  
 this.index = 0;  
 }  
  
 @Override  
 public boolean hasNext() {  
 if (index < bookShelf.getLength()) {  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 @Override  
 public Object next() {  
 Book book = bookShelf.getBookAt(index);  
 index++;  
 return book;  
 }  
}  
  
// 测试类Main  
public class Main {  
 public static void main(String[] args) {  
 BookShelf bookShelf = new BookShelf();  
 bookShelf.appendBook(new Book("Around the World in 80 Days"));  
 bookShelf.appendBook(new Book("Bible"));  
 bookShelf.appendBook(new Book("Forrest Gump"));  
 bookShelf.appendBook(new Book("Triumph"));  
  
 Iterator it = bookShelf.iterator();  
 while (it.hasNext()) {  
 Book book = (Book) it.next();  
 System.*out*.println(book.getName());  
 }  
 }  
}

运行结果：



1. 针对Iterator的例子，设计一个Specified的Iterator并运行

Code：

// 定义Aggregate接口  
interface Aggregate {  
 public abstract Iterator iterator();  
}  
  
// 定义Iterator接口  
interface Iterator {  
 public abstract boolean hasNext();  
 public abstract Object next();  
}  
  
// Book类  
class Book {  
 private String name;  
  
 public Book(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
}  
  
// BookShelf类  
class BookShelf implements Aggregate {  
 private Book[] books;  
 private int last = 0;  
  
 public BookShelf(int maxsize) {  
 this.books = new Book[maxsize];  
 }  
  
 public Book getBookAt(int index) {  
 return books[index];  
 }  
  
 public void appendBook(Book book) {  
 this.books[last] = book;  
 last++;  
 }  
  
 public int getLength() {  
 return last;  
 }  
  
 @Override  
 public Iterator iterator() {  
 return new SpecifiedBookShelfIterator(this, 1, 3); // 这里指定遍历索引1到3（不包含3）的书籍，可根据需要调整  
 }  
}  
  
// SpecifiedBookShelfIterator类，指定范围的迭代器  
class SpecifiedBookShelfIterator implements Iterator {  
 private BookShelf bookShelf;  
 private int index;  
 private int startIndex;  
 private int endIndex;  
  
 public SpecifiedBookShelfIterator(BookShelf bookShelf, int startIndex, int endIndex) {  
 this.bookShelf = bookShelf;  
 this.startIndex = startIndex;  
 this.endIndex = endIndex;  
 this.index = startIndex;  
 }  
  
 @Override  
 public boolean hasNext() {  
 if (index < endIndex && index < bookShelf.getLength()) {  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 @Override  
 public Object next() {  
 Book book = bookShelf.getBookAt(index);  
 index++;  
 return book;  
 }  
}  
  
// 测试类Main  
public class Main {  
 public static void main(String[] args) {  
 BookShelf bookShelf = new BookShelf(4);  
 bookShelf.appendBook(new Book("Around the World in 80 Days"));  
 bookShelf.appendBook(new Book("Bible"));  
 bookShelf.appendBook(new Book("Forrest Gump"));  
 bookShelf.appendBook(new Book("Triumph"));  
  
 Iterator it = bookShelf.iterator();  
 while (it.hasNext()) {  
 Book book = (Book) it.next();  
 System.*out*.println(book.getName());  
 }  
 }  
}

运行结果：

