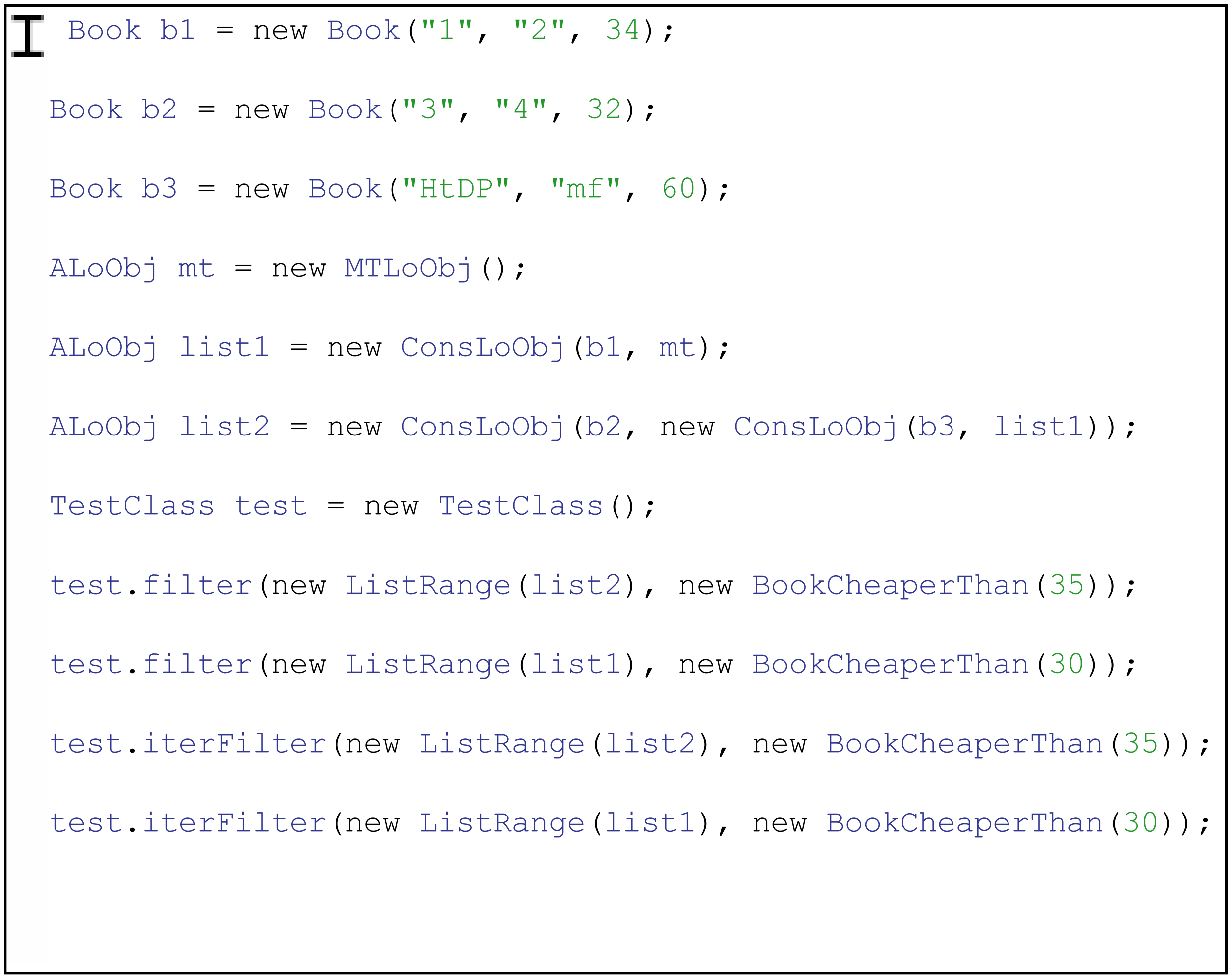
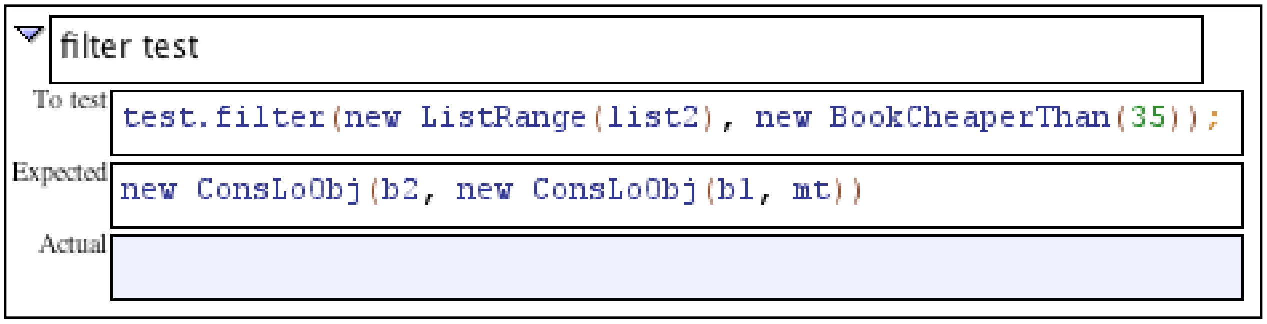
```
/////////////////////////////////////////////////////////////////////
// File lab6-part2.java

// ------------------------------------------------------------------
// interface to represent a method from object to boolean
interface IObj2Bool{
  boolean select(Object obj);
}

// ------------------------------------------------------------------
//   Functional (External) Iterator Pattern
interface IRange {

    // move the cursor one forward, assuming it is not empty
    void next();

    // select the item to which the cursor points, assuming it is not empty
    Object current();

    // test whether there are more items in the range
    boolean hasMore();
}

// ------------------------------------------------------------------
// classes to represent an arbitrary list of objects
abstract class ALoObj{ }

class MTLoObj extends ALoObj{
  MTLoObj() {}
}

class ConsLoObj extends ALoObj{
  Object first;
  ALoObj rest;

  ConsLoObj(Object first, ALoObj rest){
    this.first = first;
    this.rest = rest;
  }
}

// ------------------------------------------------------------------
// interface to represent an arbitrary stack of objects
interface IStack{

  // determine whether this is an empty stack
  boolean empty();

  // push a new object onto the stack
  void push(Object obj);

  // pop (remove) the top item off the stack
  void pop();

  // produce the top item on the stack
  Object top();

}

class ListStack implements IStack{
  ALoObj list;

  ListStack(ALoObj list){
    this.list = list;
  }

  // determine whether this is an empty stack
  boolean empty(){
    return (list instanceof MTLoObj);
  }

  // push a new object onto the stack
  void push(Object obj){
    list = new ConsLoObj(obj, list);
  }

  // pop (remove) the top item off the stack
  void pop(){
    list = ((ConsLoObj)list).rest;
  }

  // produce the top item on the stack
  Object top(){
    return ((ConsLoObj)list).first;
  }
}

// ------------------------------------------------------------------
//   Functional (External) Iterator Pattern:
//        iterator for a list of objects

class ListRange implements IRange {
    // ---------------------------------------------------------
    // Member data
    ALoObj ptr; /* reference to this list */

    //---------------------------------------------------------
    // Constructor
    ListRange(ALoObj aList) { this.ptr = aList; }

    //---------------------------------------------------------
    // Methods to implement the IRange interface
    void next() {
        this.ptr = ((ConsLoObj)this.ptr).rest;
    }

    Object current() {
        return ((ConsLoObj)this.ptr).first;
    }

    boolean hasMore() {
        return (this.ptr instanceof ConsLoObj);
    }
}


// ------------------------------------------------------------------
// objects to keep in the list
class Book {
  String title;
  String author;
  int price;

  Book(String title, String author, int price) {
    this.title = title;
    this.author = author;
    this.price = price;
  }

}


// ------------------------------------------------------------------
// the client class that uses the list of objects, the iterator, and the
IObj2Bool interface
class TestClass{
```

```
TestClass(){}

// externally defined recursive filter
ALoObj filter(IRange it, IObj2Bool io2b){

  // not empty?
  if (it.hasMore()){
    // remember local value and advance the iterator
    Object obj = it.current();
    it.next();

    // select this item?, add it and go on
    if (io2b.select(obj))
      return new ConsLoObj(obj, filter(it, io2b));

    // not selected, go on
    else
      return filter(it, io2b);
  }
  // empty clause
  else
    return new MTLoObj();
}

// external iterative filter
ALoObj iterFilter(IRange it, IObj2Bool io2b){

  // empty clause
  ALoObj result = new MTLoObj();

  // traverse the list
  for (IRange r = it; r.hasMore(); r.next()){
    if (io2b.select(r.current()))
      result = new ConsLoObj(r.current(), result);
  }

  // return the result - note that it is in reverse order
  return result;
}

// list reversal using the ListStack and the ListRange iterator
ALoObj reverse(IRange it){
  // start with an empty stack
  ListStack s = new ListStack(new MTLoObj());

  // push each new item onto the stack
```

```
    for (IRange r = it; r.hasMore(); r.next()){
      s.push(r.current());
    }

    // return the contents of the stack
    return ((ListStack)s).list;
  }

}

// book selector by price: cheaper than given price
class BookCheaperThan implements IObj2Bool{
  int price;

  BookCheaperThan(int price){
    this.price = price;
  }

  // select Book object with cheaper price that this.price
  boolean select(Object obj){
    return (((Book)obj).price) < this.price;
  }
}
```

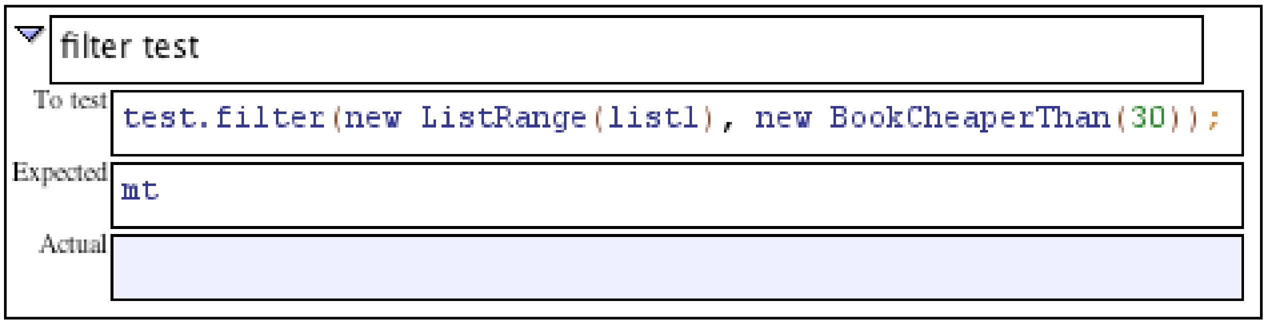


```
Book b1 = new Book("1", "2", 34);

Book b2 = new Book("3", "4", 32);

Book b3 = new Book("HtDP", "mf", 60);

ALoObj mt = new MTLoObj();

ALoObj list1 = new ConsLoObj(b1, mt);

ALoObj list2 = new ConsLoObj(b2, new ConsLoObj(b3, list1));

TestClass test = new TestClass();

test.filter(new ListRange(list2), new BookCheaperThan(35));

test.filter(new ListRange(list1), new BookCheaperThan(30));

test.iterFilter(new ListRange(list2), new BookCheaperThan(35));

test.iterFilter(new ListRange(list1), new BookCheaperThan(30));
```



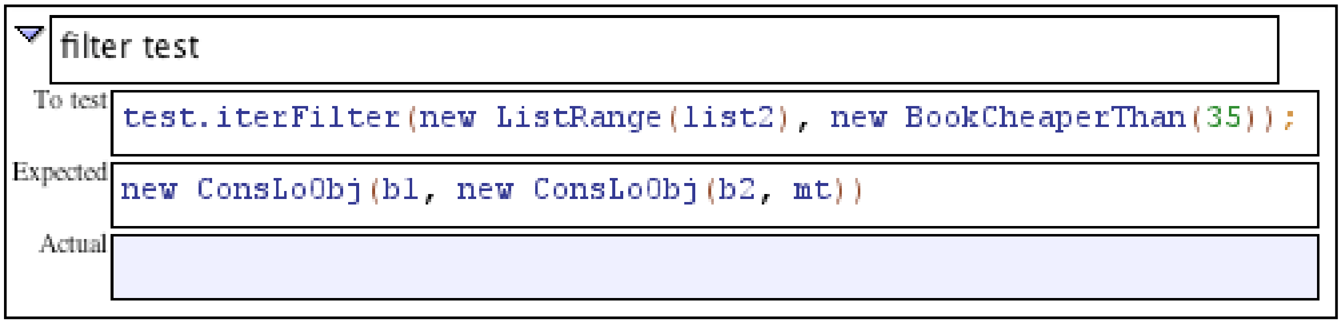
filter test

To test
```
test.filter(new ListRange(list2), new BookCheaperThan(35));
```
Expected
```
new ConsLoObj(b2, new ConsLoObj(b1, mt))
```
Actual

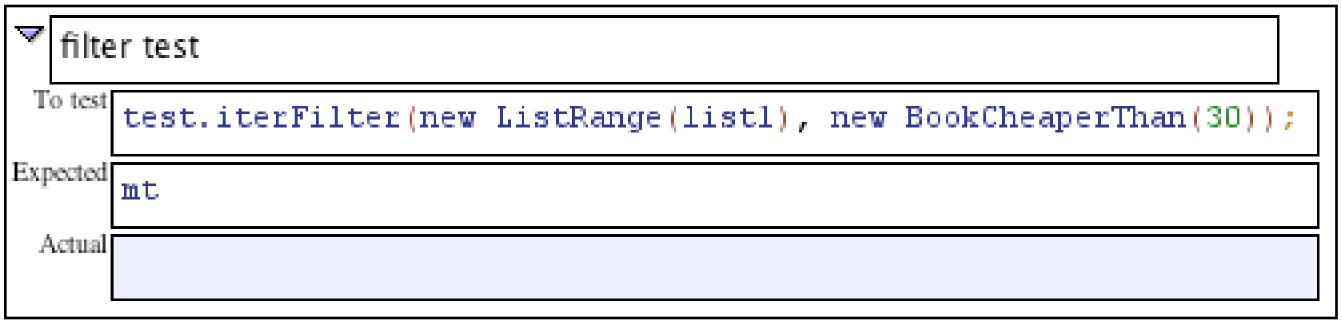


filter test

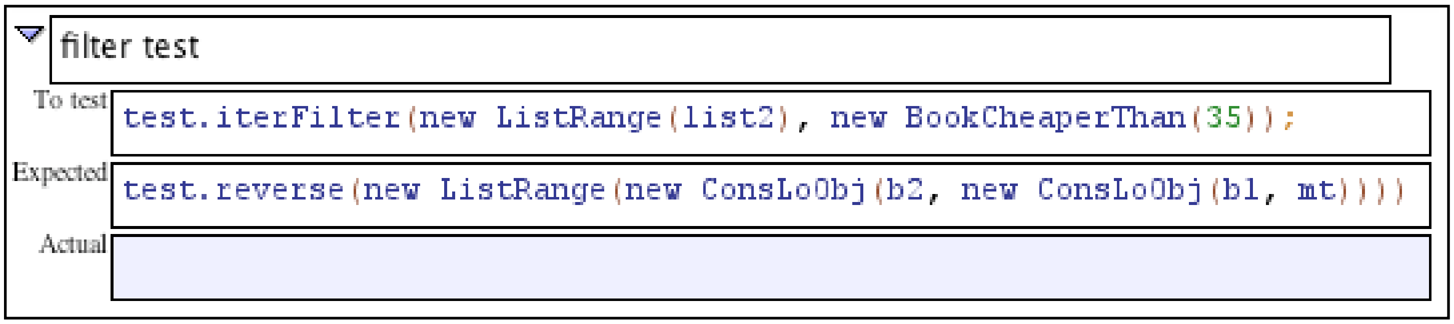
To test
```
test.filter(new ListRange(list1), new BookCheaperThan(30));
```
Expected
```
mt
```
Actual


filter test

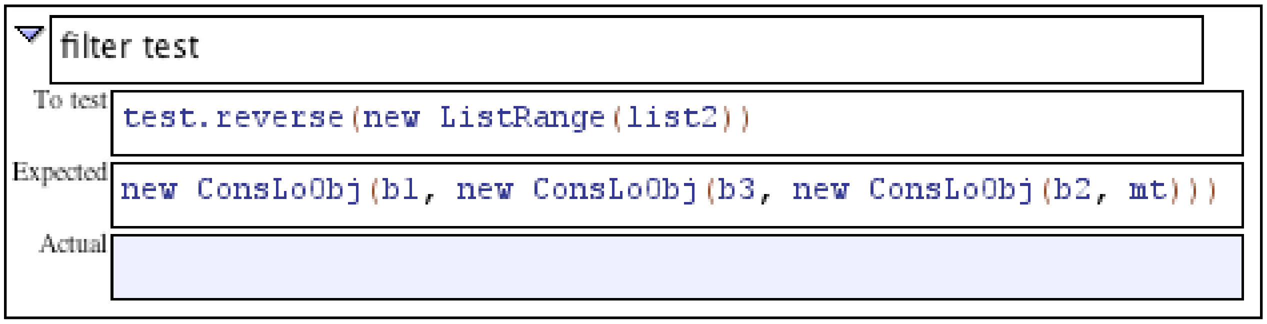
To test
```
test.iterFilter(new ListRange(list2), new BookCheaperThan(35));
```
Expected
```
new ConsLoObj(b1, new ConsLoObj(b2, mt))
```
Actual


filter test

To test
```
test.iterFilter(new ListRange(list1), new BookCheaperThan(30));
```
Expected
```
mt
```
Actual


filter test

To test
```
test.iterFilter(new ListRange(list2), new BookCheaperThan(35));
```
Expected
```
test.reverse(new ListRange(new ConsLoObj(b2, new ConsLoObj(b1, mt))))
```
Actual


filter test

To test
```
test.reverse(new ListRange(list2))
```
Expected
```
new ConsLoObj(b1, new ConsLoObj(b3, new ConsLoObj(b2, mt)))
```
Actual