

*Reference information about some standard Java library classes appears on the last pages of the test. You can tear off these pages for easier reference during the exam if you like.*

**Question 1.** (2 points) When a method is called we often want to check that its precondition is met before doing anything else. One way to handle this is to use an `assert` statement; another way is to throw an exception. When would it be appropriate to throw an exception instead of using `assert`?

**Question 2.** (2 points) After we are done using a stream, the stream should be closed, either automatically when the program finishes (i.e., method `main` terminates), or explicitly by executing the `close()` method on the stream. What is the purpose of closing a stream – i.e., what does the close operation do?

The next question involves the following code, which you can remove from the test for reference if you wish.

```
public class TestQuestion {

    public void a(int n) {
        System.out.println("start a");
        b(n);
        System.out.println("end a");
    }

    public void b(int n) {
        System.out.println("start b");
        try {
            c(n);
        } catch (NullPointerException e) {
            System.out.println("Caught NullPointerException in b");
            if (n%2 == 0) {
                throw new IndexOutOfBoundsException("from b");
            }
        }
        System.out.println("end b");
    }

    public void c(int n) {
        System.out.println("start c");
        if (n > 10) {
            throw new NullPointerException("from c");
        }
        System.out.println("end c");
    }
}
```

**Question 3.** (8 points) What happens when each of the following sets of statements are executed using the code in the class on the previous page? You should indicate what output is produced and show where exceptions are thrown and caught, if this happens.

(a) `TestQuestion q = new TestQuestion();`  
`q.a(10);`

(b) `TestQuestion q = new TestQuestion();`  
`q.a(12);`

**Question 4.** (2 points) The `iterator()` method in collection classes like our `SimpleArrayList` returns a new instance of a class that implements `Iterator`, which includes methods `hasNext()` and `next()`. Why is an iterator usually implemented as a separate class, instead of just including methods `hasNext()` and `next()` as methods in the collection itself? In other words, what advantage, if any, is there to having an iterator as a separate object?

**Question 5.** (10 points) (a) (8 points) We would like to add a method to our `SimpleArrayList` class to remove an item at a given position from the list. More specifically, complete the definition of the method `remove`, below, so that

```
remove(int pos)
```

removes the list element at the given position `pos`, or throws an `IndexOutOfBoundsException` if `pos` is negative or greater than or equal to the current size of the list. Recall that the state of a `SimpleArrayList` is represented as follows:

```
private Object[] items; // items in this list are stored in
private int size;      // items[0..size-1]
```

After removing the requested item, the list should be properly adjusted to reflect the class invariant in the comments above, and the `items` array should not contain any extra references to objects other than the ones necessary to represent the contents of the list.

```
public void remove(int pos) {
```

```
}
```

(b) (2 points) Do we need to add “throws `IndexOutOfBoundsException`” to the heading of this method to get it to compile? Why or why not?

**Question 6.** (18 points) In this question we would like to read a file containing definitions of words and then be able to find all the words whose definitions *contain* a particular word. The format of the input file consists of sequences of words followed by their definitions, separated by blank lines. Here is an example of a sample input file containing three definitions.

```
frog
a frog is a smooth or
slimy creature with two
bulging eyes and webbed
hind feet

toad
a kind of frog with a stubby body and
short hind legs; generally with warts
and dry skin

newt
an amphibian that is sometimes interested
in politics
```

(a) (10 points) Complete the definition of method `readDictionary` on the next page so that it reads the contents of the `BufferedReader` stream in one line at a time (using the `readLine()` method), and stores the words and their definitions in the `HashMap` dictionary. Your code should store each definition as a single long `String` with one extra blank between each line from the original input file, and with no other characters like newlines separating the input lines.

You should make the following assumptions to simplify the task:

- There are no blank or empty lines before the first word/definition pair in the file,
- Every definition contains at least one line,
- Every definition, including the last one, is followed by exactly one line containing a zero-character string (“”), and
- No word is defined more than once in the input file.

If an exception occurs while the file is being read, you should print an appropriate message and stop.

[Turn to the next page to write your answer. Also, remember there is some reference information on the last pages of the test that you might find useful.]

**Question 6(a).** (10 points)

```
/** Read words and definitions from stream in and store
 * the definitions in the map dictionary using the
 * words as the keys */
public void readDictionary(BufferedReader in,
                          HashMap dictionary) {

}
```

**Question 6 (b)** (8 points) Complete the definition of the following method so it prints to `System.out` the words (keys) in the `HashMap` `dictionary` whose *definition* (not key) contains the string `target`. So, for example, if we execute `printWords("frog", dictionary)`, where `dictionary` has been initialized with the definitions given at the beginning of this question, the method should print the words "frog" and "toad", since the word "frog" occurs in the definitions of those two words. Your solution only needs to detect exact matches, i.e., if the word is "frog", you only need to find definitions that contain "frog". You can ignore definitions that contain "Frog", "FROG", and similar words as long if they don't contain the string "frog".

```
/** Print on System.out all words (keys) in dictionary
 * whose definitions contain the string target */
public void printWords(String target, HashMap dictionary) {
```

```
}
```

## Java Reference Information

Feel free to detach these pages and use them for reference as you work on the exam.

class **BufferedReader**

String readLine()                      Return next line from input stream, or null if no more input. Can throw IOException.

class **PrintWriter**

void print(arg)                      Print arg to the `PrintWriter` stream. The parameter can be any type

void println()                      Terminate the current output line and move to the beginning of the next. line

void println(arg)                      Print arg, then advance to the beginning of the next line

class **String**

All of the search methods in class `String` return -1 if the item is not found

int length()                      length of this string

int indexOf(char ch)                      first position of ch

int indexOf(char ch, int start)                      first position of ch starting from start

int indexOf(String str)                      first position of str

int indexOf(String str, int start)                      first position of str starting from start

int lastIndexOf(char ch)                      last position of ch

int lastIndexOf(char ch, int start)                      last position of ch searching backward from start

int lastIndexOf(String str)                      last position of str

int lastIndexOf(String str, int start)                      last position of str searching backward from start

String substring(int start)                      substring of this string from position start to the end

String substring(int start, end)                      substring of this string from position start to end-1

String trim()                      copy of this string with leading and trailing whitespace deleted



All **Collection** interfaces (**List**, **Set**) and classes (**ArrayList**, **LinkedList**, **HashSet**, **TreeSet**)

```

boolean add(Object obj)
boolean addAll(Collection other)
void clear()
boolean contains(Object obj)
Iterator iterator()
boolean remove(Object obj)
int size()
Object[] toArray() // return an array containing all the
                  // elements in this collection

```

In addition, all **Collection** classes provide a constructor that takes another **Collection** as a parameter and creates a new collection whose initial contents are copied from that parameter. (i.e., `public ArrayList(Collection c)`, and similarly for the other classes.)

Additional methods in **List**, **ArrayList**, **LinkedList**

```

add(int position, Object obj)
remove(int position)

```

**Map**, **HashMap**, **TreeMap**

```

Object put(Object key, Object value)
Object get(Object key)
Object remove(Object key)
Set keySet()
Collection values()
int size()

```

## **arrays**

If `a` is a Java array, `a.length` is the number of elements in that array.

If `m` is a 2-dimensional Java array, `m[k]` refers to row `k` of the array, and `m[k].length` is the length of that row (which is the same for all rows in a normal, rectangular array).

## **Exceptions**

Some standard exceptions that might be useful: `IllegalArgumentException`, `IndexOutOfBoundsException`, `NoSuchElementException`, `NullPointerException`