PRACTICE EXAM 1 – FREE-RESPONSE QUESTIONS

Question 1.

This question involves writing a class, writing a class that extends that class, and writing a method that accepts the parent class type.

(a) A *number group* represents a group of integers defined in some way. It could be empty, or it could contain one or more integers.

Write a class named `NumberGroup` that represents a group of integers. The class should have a single `contains` method that determines if a given integer is in the group. For example, if `group1` is of type `NumberGroup`, and it contains only the two numbers `-5` and `3`, then `group1.contains(-5)` would return `true`, and `group1.contains(2)` would return `false`.

Write the complete `NumberGroup` class.

(b) A *range* represents a number group that contains all (and only) the integers between a minimum value and

a maximum value, inclusive.

Write the `Range` class, which extends `NumberGroup`. The `Range` class represents the group of `int` values that range from a given minimum value up through a given maximum value, inclusive. For example, the declaration

```
NumberGroup range1 = new Range(-3, 2);
```

represents the group of integer values -3, -2, -1, 0, 1, 2.

Write the complete `Range` class. Include all necessary instance variables and methods as well as a constructor that takes two `int` parameters. The first parameter represents the minimum value, and the second parameter represents the maximum value of the range. You may assume that the minimum is less than or equal to the maximum.

(c) The `MultipleGroups` class (not shown) represents a collection of `NumberGroup` objects and also extends the class `NumberGroup`. The `MultipleGroups` class stores the number groups in the instance variable `groupList` (shown below), which is initialized in the constructor.

```
private List<NumberGroup> groupList;
```

Write the `MultipleGroups` method `contains`, which overrides the `contains` method in the parent class `NumberGroup`. This `contains` method takes an integer and returns `true` if and only if the integer is contained in one or more of the number groups in `groupList`.

For example, suppose `multiple1` has been declared as an instance of `MultipleGroups` and consists of the three ranges created by the calls `new Range(5, 8)`, `new Range(10, 12)`, and `newRange(1,6)`. The following table shows the results of several calls to `contains`.

Call	Result
`multiple1.contains(2)`	`true`
`multiple1.contains(9)`	`false`
`multiple1.contains(6)`	`true`

Question 2.

A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the *crossword labeling rule*.

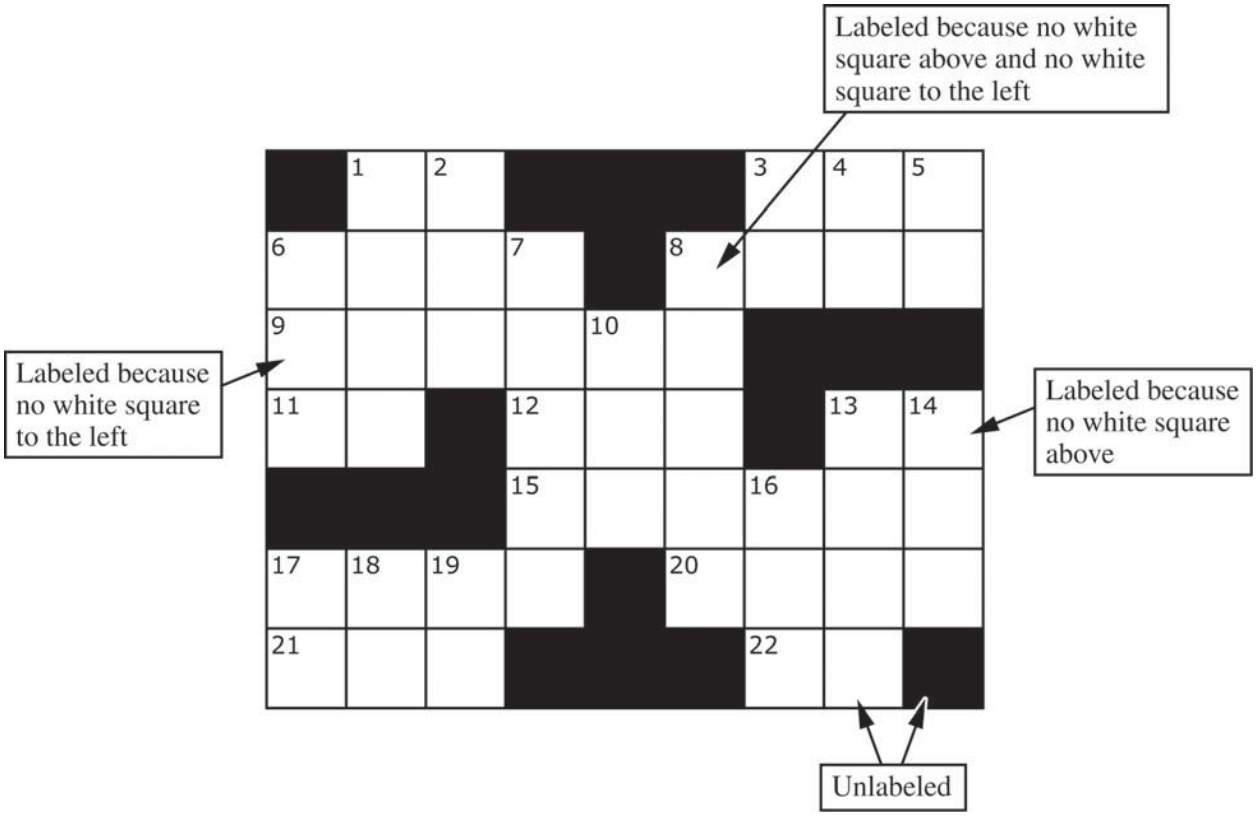
The crossword labeling rule identifies squares to be labeled with a positive number as follows.

A square is labeled with a positive number if and only if

- the square is white and
- the square does not have a white square immediately above it, or it does not have a white square immediately to its left, or both.

The squares identified by these criteria are labeled with consecutive numbers in row-major order, starting at 1.

The following diagram shows a crossword puzzle grid and the labeling of the squares according to the crossword labeling rule.



PRACTICE EXAM 1 – FREE-RESPONSE QUESTIONS

This question uses two classes, a `Square` class that represents an individual square in the puzzle and a `Crossword` class that represents a crossword puzzle grid. A partial declaration of the `Square` class is shown below.

```
public class Square
{
    /* Constructs one square of a crossword puzzle grid.
     * Postcondition:
     * – The square is black if and only if isBlack is true.
     * – The square has number num.
     */
    public Square(boolean isBlack, int num)
    { /* to be implemented in part (a) */ }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

A partial declaration of the `Crossword` class is shown below. You will implement one method and the constructor in the `Crossword` class.

```
public class Crossword
{
    /** Each element is a Square object with a color (black or white) and a number.
     * puzzle[r][c] represents the square in row r, column c.
     * There is at least one row in the puzzle.
     */
    private Square[][] puzzle;


    /** Constructs a crossword puzzle grid.
     * Precondition: There is at least one row in blackSquares.
     * Postcondition:
     * – The crossword puzzle grid has the same dimensions as blackSquares.
     * – The Square object at row r, column c in the crossword puzzle grid is black
     *     if and only if blackSquares[r][c] is true.
     * – The squares in the puzzle are labeled according to the crossword labeling rule.
     */
    public Crossword(boolean[][] blackSquares)
    { /* to be implemented in part (b) */ }


    /** Returns true if the square at row r, column c should be labeled with a positive
                  number; false otherwise.
     * The square at row r, column c is black if and only if blackSquares[r][c] is
       true.
     * Precondition: r and c are valid indexes in blackSquares.
     */
    private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
    { /* implementation not shown */ }


    // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the `Square` class. The constructor accepts a `boolean` and `int` and initializes the appropriate instance variables. Additionally, write the methods `isBlack` and `getNum`. `isBlack` returns a `boolean` (black: `true`, white: `false`), and `getNum` returns an `int` that is used to label the square.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Write the `Square` class below.

```
public class Square
{
```

(b) Write the `Crossword` constructor. The constructor should initialize the crossword puzzle grid to have the same dimensions as the parameter `blackSquares`. Each element of the puzzle grid should be initialized with a reference to a `Square` object with the appropriate color and number. The number is positive if the square is labeled and 0 if the square is not labeled.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `toBeLabeled` works as specified, regardless of what you wrote in part (a). You must use `toBeLabeled` appropriately to receive full credit.

Complete the `Crossword` constructor below.

```
/** Constructs a crossword puzzle grid.
  * Precondition: There is at least one row in blackSquares.
  * Postcondition:
  * – if and only if blackSquares[r][c] is true.
  * – The crossword puzzle grid has the same dimensions as blackSquares.
  * – The Square object at row r, column c in the crossword puzzle grid is black
  * – The squares in the puzzle are labeled according to the crossword labeling rule.
  */
public Crossword(boolean[][] blackSquares)
```

Question 3.

This question involves identifying and processing the binary digits of a non-negative integer. The declaration of the `BinaryDigits` class is shown below. You will write the constructor and one method for the `BinaryDigits` class.

```
public class BinaryDigits
{

    /** The list of binary digits that represent the number used to construct this object.
     *  The digits appear in the list in the order in which they appear in the binary representation
     *  of the number.
     */
    private ArrayList<Integer> digitList;


    /** Constructs a Digits object that represents num.
     *  Precondition: num >= 0
     */
    public BinaryDigits(int num)
    { /* to be implemented in part (a) */ }


    /** Returns true if the digits in this Digits object contain a sequence of three digits with value 1;
     *  false otherwise.
     */
    public boolean hasOnesGroup()
    { /* to be implemented in part (b) */ }
}
```

(a) Write the constructor for the `BinaryDigits` class. The constructor initializes and fills `digitList` with the binary digits from the non-negative integer `num`. The elements in `digitList` must be `Integer` objects representing single digits (1 or 0).

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete the `BinaryDigits` constructor below.

```
/** Constructs a Digits object that represents num.
 * Precondition: num >= 0
 */
public BinaryDigits(int num)
```

(b) Write the `Digits` method `hasOnesGroup`. The method returns `true` if the elements of `digitList` contain 3 or more contiguous elements of value 1; otherwise, it returns `false`.

The following table shows the results of several calls to `hasOnesGroup`.

Method call	Binary representation	Value returned
`new BinaryDigits(14).hasOnesGroup ()`	1110	`true`
`new BinaryDigits(55).hasOnesGroup ()`	110111	`true`
`new BinaryDigits(107).hasOnesGroup ()`	1101011	`false`
`new BinaryDigits(21).hasOnesGroup ()`	10101	`false`
`new BinaryDigits(7).hasOnesGroup ()`	111	`true`

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `hasOnesGroup` below.

```
/** Returns true if the digits in this BinaryDigits object are in strictly increasing order;
  * false otherwise.
  */
public boolean hasOnesGroup ()
```

Question 4.

This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
public class WordPair
{
      /** Constructs a WordPair object. */
      public WordPair(String first, String second)
      { /* implementation not shown */ }

      /** Returns the first string of this WordPair object. */
      public String getFirst()
      { /* implementation not shown */ }

      /** Returns the second string of this WordPair object. */
      public String getSecond()
      { /* implementation not shown */ }
}
```

You will implement the constructor and another method for the following `WordPairList` class.

```
public class WordPairList
{
      /** The list of word pairs, initialized by the constructor. */
      private ArrayList<WordPair> allPairs;

      /** Constructs a WordPairList object as described in part (a).
       * Precondition: words.length >= 2
       */
      public WordPairList(String[] words)
      { /* to be implemented in part (a) */ }

      /** Returns a list of repeated words as described in part (b). */
      public ArrayList<String> repeatedWords()
      { /* to be implemented in part (b) */ }
}
```

(a) Write the constructor for the `WordPairList` class. The constructor takes an array of strings `words` as a parameter and initializes the instance variable `allPairs` to an `ArrayList` of `WordPair` objects.

A `WordPair` object consists of a word from the array paired with a word that appears later in the array. The `allPairs` list contains `WordPair` objects `(words[i], words[j])` for every `i` and `j`, where 0 £ `i < j < words.length` . Each `WordPair` object is added exactly once to the list.

The following examples illustrate two different `WordPairList` objects.

Example 1

```
String[] wordNums = {"one", "two", "three"};
WordPairList exampleOne = new WordPairList(wordNums);
```

After the code segment has executed, the `allPairs` instance variable of `exampleOne` will contain the following `WordPair` objects in some order.

```
("one", "two"), ("one", "three"), ("two", "three")
```

Example 2

```
String[] phrase = {"the", "more", "the", "merrier"};
WordPairList exampleTwo = new WordPairList(phrase);
```

After the code segment has executed, the `allPairs` instance variable of `exampleTwo` will contain the following `WordPair` objects in some order.

```
("the", "more"), ("the", "the"), ("the", "merrier"),
("more", "the"), ("more", "merrier"), ("the", "merrier")
```

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public ArrayList<String> repeatedWords()
```

Complete the `WordPairList` constructor below.

```
/** Constructs a WordPairList object as described in part (a).
 * Precondition: words.length >= 2
 */
public WordPairList(String[] words)
```

(b) Write the `WordPairList` method `repeatedWords`. This method returns an `ArrayList<String>` containing the list of words that are repeated.

For example, the following code segment creates a `WordPairList` object. `String[] moreWords = {"the", "red", "fox", "the", "red"};`

```
WordPairList exampleThree = new WordPairList(moreWords);
```

After the code segment has executed, the `allPairs` instance variable of `exampleThree` will contain the following `WordPair` objects in some order. The pairs in which the first string matches the second string are shaded for illustration.

```
("the", "red"), ("the", "fox"), ("the", "the"),
("the", "red"), ("red", "fox"), ("red", "the"),
("red", "red"), ("fox", "the"), ("fox", "red"),
("the", "red")
```

The call `exampleThree.numMatches()` should return an `ArrayList<String>` containing the following `Strings`:

```
"the", "red"
```

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public ArrayList<String> repeatedWords()
```

Complete method `repeatedWords` below.

```
/** Returns a list of repeated words as described in part (b). */
public ArrayList<String> repeatedWords()
```