Chapter	Section	No
1	1	1

Question

Define an interface named `Resizable` with just one abstract method, `resize`, that is a **`void`** method with no parameter.

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 1
//File: \KW\CH01\Resizable.java startLine: 0 endLine 14
package KW.CH01;

/**
 *  Interface to define the method resize
 */
public interface Resizable {

    /**
     *  Method to resize the object
     */
    void resize();
}
```

Chapter	Section	No
1	1	3

Question

Write a Javadoc comment for the following method of class `Person`. Provide preconditions and postconditions if needed.

```
Public void changeLastName(boolean justMarried, String newLast) {
    if (justMarried)
        lastName = newLast;
}
```

Solution

```
//Solution to programming exercise 3, Section 1, Chapter 1
//File: \KW\CH01\Person.java startLine: 199 endLine 213
    /**
     * Method to set the last name of this Person to a new
     * value provided that justMarried is true.
     *
     * @param justMarried true if this Person's name is to be changed
     *        param newLast The new last name if justMarried is true
     * @param newLast DOCUMENT ME!
     *
     * @post lastName is equal to newLast if justMarried is true
     *       otherwise no change is made to this Person
     */
```

Chapter	Section	No
1	2	1

Question

Write accessor and modifier methods for class `Computer`.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 1
//File: \KW\CH01\Computer.java startLine: 60 endLine 90
    public String getManufacturer() {
        return manufacturer;
    }

    public String getProcessor() {
        return processor;
    }

    public void setManufacurer(String man) {
        manufacturer = man;
    }

    public void setProcessor(String processor) {
        this.processor = processor;
    }

    public void setRamSize(int ram) {
        ramSize = ram;
    }

    public void setDiskSize(int disk) {
```

```
        diskSize = disk;
    }

    public void setProcessorSpeed(double procSpeed) {
        processorSpeed = procSpeed;
    }
```

Chapter	Section	No
1	3	1

Question

Write constructors for both classes that allow you to specify only the processor, RAM size, and disk size.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 1
//File: \KW\CH01\Notebook.java startLine: 34 endLine 39
    public Notebook(String processor, double ram, int disk) {
        this("Default", processor, ram, disk, 2.5, 17, 5.5);
    }
//Solution to programming exercise 1, Section 3, Chapter 1
//File: \KW\CH01\Computer.java startLine: 37 endLine 43
    public Computer(String processor, double ram, int disk) {
        this("Default", processor, ram, disk, 2.5);
    }
```

Chapter	Section	No
1	3	3

Question

Complete the accessor and modifier methods for class `Notebook`.

Solution

```
//Solution to programming exercise 3, Section 3, Chapter 1
//File: \KW\CH01\Notebook.java startLine: 59 endLine 62
    // See the solution for 1.2.2
```

Chapter	Section	No
1	4	1

Question

Write class `Vegetable`. Assume that a vegetable has three double constants: `VEG_PROTEIN_CAL`, `VEG_FAT_CAL`, and `VEG_CARBO_CAL`. Compute the fat percentage as `VEG_FAT_CAL` divided by the sum of all the constants.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 1
//File: \KW\CH01\Vegetable.java startLine: 0 endLine 51
package KW.CH01;

/**
 * Class to represent a Vegetable
 */
public class Vegetable extends Food {

    /** Calories from protein */
    private static final double VEG_PROTEIN_CAL = 0.35;
    /** Calories from fat */
    private static final double VEG_FAT_CAL = 0.15;
    /** Calories from carbohydrates */
    private static final double VEG_CARBO_CAL = 0.50;
    /** The name of the vegetable */
    private String name;

    /**
     * Constructor
     * @param name The name of the vegetable
     */
    public Vegetable(String name) {
        this.name = name;
        setCalories(VEG_PROTEIN_CAL + VEG_FAT_CAL + VEG_CARBO_CAL);
    }

    /**
     * Calculates the percent of protein in a Food object.
     * @return The percentage of protein
     */
    public double percentProtein() {
```

```
        return VEG_PROTEIN_CAL / getCalories();
    }

    /**
     * Calculates the percent of fat in a Food object.
     * @return The precentage of fat
     */
    public double percentFat() {
        return VEG_FAT_CAL / getCalories();
    }

    /**
     * Calculates the percent of carbohydrates in a Food object.
     * @return The percentage of carbohydrates
     */
    public double percentCarbohydrates() {
        return VEG_CARBO_CAL / getCalories();
    }
}
```

Chapter Section No
1 5 1

Question

Write an equals method for class `Computer` (Listing 1.2).

Solution

```
//Solution to programming exercise 1, Section 5, Chapter 1
//File: \KW\CH01\Computer.java startLine: 130 endLine 164
    /**
     * Determine if this Computer is equal to the other
     * object
     *
     * @param obj The object to compare this Computer to
     *
     * @return true If the other object is of type Computer and all
     *         data fields are equal
     */
    @Override
    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }

        if (obj == null) {
            return false;
        }

        if (obj.getClass() == this.getClass()) {
            Computer other = (Computer) obj;

            return getManufacturer().equals(other.getManufacturer())
                    && getProcessor().equals(other.getProcessor())
                    && (getRamSize() == other.getRamSize())
                    && (getDiskSize() == other.getDiskSize())
                    && (getProcessorSpeed() == other.getProcessorSpeed());
        } else {
            return false;
        }
    }
```

Chapter Section No
1 5 3

Question

Write an equals method for the following class. What other equals methods should be defined?

```
public class Airplane {
// Data Fields
Engine eng;
Rudder rud;
Wing[] wings = new Wing[2];
...
}
```

Solution

```
//Solution to programming exercise 3, Section 5, Chapter 1
//File: \KW\CH01\Airplane.java startLine: 11 endLine 34
    @Override
    public boolean equals(Object obj) {
        if (obj == this) {
```

```
            return true;
        }

        if (obj == null) {
            return false;
        }

        if (obj.getClass() == this.getClass()) {
            Airplane other = (Airplane) obj;

            return eng.equals(other.eng) && rud.equals(other.rud)
                    && wings[0].equals(other.wings[0])
                    && wings[1].equals(other.wings[1]);
        } else {
            return false;
        }
    }
    // equals methods need to be defined for classes Engine, Rudder, and Wing
```

Chapter	Section	No
1	8	1

Question

Write class `Circle`.

Solution

```
//Solution to programming exercise 1, Section 8, Chapter 1
//File: \KW\CH01\Circle.java startLine: 0 endLine 76
package KW.CH01;

import java.util.Scanner;

/**
 * Represents a circle.
 * Extends Shape.
 */
public class Circle extends Shape {
    // Data Fields

    /** The radius of the circle */
    private double radius = 0;

    // Constructors
    /** Constructs a default Circle */
    public Circle() {
        super("Circle");
    }

    /**
     * Constructs a circle of the specified size.
     * @param radius the radius
     * @param height the height
     */
    public Circle(double radius) {
        super("Circle");
        this.radius = radius;
    }

    // Methods
    /**
     * Get the radius.
     * @return The width
     */
    public double getRadius() {
        return radius;
    }

    /**
     * Compute the area.
     * @return The area of the circle
     */
    @Override
    public double computeArea() {
        return Math.PI * radius * radius;
    }

    /**
     * Compute the perimeter.
     * @return The perimeter of the circle
     */
    @Override
```

```
    public double computePerimeter() {
        return 2 * Math.PI * radius;
    }

    /** Read the attributes of the circle. */
    @Override
    public void readShapeData() {
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the radius of the Circle");
        radius = in.nextDouble();
    }

    /**
     * Create a string representation of the circle.
     * @return A string representation of the circle
     */
    @Override
    public String toString() {
        return super.toString() + ": radius is " + radius;
    }
}
```

Chapter Section No
2 1 1

Question

Write the following static method:

```
/** Replaces each occurrence of oldItem in aList with newItem. */
public static void replace(ArrayList<String> aList, String oldItem,
    String newItem)
```

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 2
//File: \KW\CH02\Exercises.java startLine: 9 endLine 28
  /**
   * Method to replace each occurence of oldItem with newItem
   * in an ArrayList<String>
   * @param aList The arraylist in which items are to be replaced
   * @param oldItem The item to be replaced
   * @param newItem The item to rplace oldItem
   * @post All occurences of oldItem have been replaced with newItem
   */
  public static void replace(ArrayList<String> aList,
      String oldItem,
      String newItem) {
    int index = aList.indexOf(oldItem);
    while (index != -1) {
      aList.set(index, newItem);
      index = aList.indexOf(oldItem);
    }
  }
```

Chapter Section No
2 2 1

Question

Write a method `addOrChangeEntry` for a class that has a data field `theDirectory` (type `ArrayList<DirectoryEntry>`) where class `DirectoryEntry` is described just before the exercises. Assume class `DirectoryEntry` has an accessor method `getNumber` and a setter method `setNumber`.

```
/** Add an entry to theDirectory or change an existing entry.
    @param aName The name of the person being added or changed
    @param newNumber The new number to be assigned
    @return The old number, or if a new entry, null
*/
public String addOrChangeEntry(String aName, String newNumber) {
```

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 2
//File: \KW\CH02\PhoneDirectory.java startLine: 17 endLine 41
    /**
```

```
     * Add an entry or change an existing entry.
     * @param name The name of the person being added or changed.
     * @param newNumber The new number to be assigned.
     * @return The old number, or if a new entry, null.
     */
    public String addOrChangeEntry(String name,
            String newNumber) {
        int index = 0;
        String oldNumber = null;
        while (index < theDirectory.size()
                && !theDirectory.get(index).getName().equals(name)) {
            index++;
        }
        if (index < theDirectory.size()) {
            oldNumber = theDirectory.get(index).getNumber();
            theDirectory.get(index).setNumber(newNumber);
        } else {
            theDirectory.add(new DirectoryEntry(name, newNumber));
        }
        return oldNumber;
    }
```

Chapter	Section	No
2	3	1

Question

Implement the `indexOf` method of the `KWArrayList<E>` class.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 2
//File: \KW\CH02\KWArrayList.java startLine: 134 endLine 154
    /**
     * Returns the index of the first occurence of the specified element
     * in this list, or -1 if this list does not contain the element
     * @param item The object to search for
     * @returns The index of the first occurence of the specified item
     *          or -1 if this list does not contain the element
     */
    public int indexOf(Object item) {
        for (int i = 0; i < size; i++) {
            if (theData[i] == null && item == null) {
                return i;
            }
            if (theData[i].equals(item)) {
                return i;
            }
        }
        return -1;
    }
```

Chapter	Section	No
2	4	1

Question

Write a program that compares the values of `y1` and `y2` in the following expressions for values of n up to 100 in increments of 10. Does the result surprise you?

```
y1 = 100 * n + 10
y2 = 5 * n * n + 2
```

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 2
//File: \KW\CH02\Question_2_4_1.java startLine: 1 endLine 16

public class Question_2_4_1 {

    public static void main(String[] args) {
        int n;
        int y1;
        int y2;
        System.out.printf("%10s%10s%10s%n", "n", "y1", "y2");
        for (n = 0; n <= 100; n += 10) {
            System.out.printf("%10d%10d%10d%n", n, 100 * n + 10, 5 * n * n + 2);
        }
    }
}
```

Chapter	Section	No

Chapter	Section	No
2	5	1

Question

Write a `remove` method that removes the item at a specified index.

Solution

```
//Solution to programming exercise 1, Section 5, Chapter 2
//File: \KW\CH02\SingleLinkedList.java startLine: 176 endLine 198
    /**
     * Remove the item at the specified position in the list. Shifts
     * any squsequent items to the left (subtracts one from their
     * indicies). Returns the tiem that was removed.
     * @param index The index of the item to be removed
     * @returns The item that was at the specified position
     * @throws IndexOutOfBoundsException if the index is out of range
     */
    public E remove(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException(Integer.toString(index));
        }
        Node<E> removedNode = null;
        if (index == 0) {
            return removeFirst();
        } else {
            Node<E> node = getNode(index - 1);
            return removeAfter(node);
        }
    }
```

Chapter	Section	No
2	5	3

Question

Write method remove for class `SingleLinkedList<E>`:

```
/** Remove the first occurrence of element item.
    @param item The item to be removed
    @return true if item is found and removed; otherwise, return false.*/
public boolean remove(E item)
```

Solution

```
//Solution to programming exercise 3, Section 5, Chapter 2
//File: \KW\CH02\SingleLinkedList.java startLine: 227 endLine 252
    /**
     * Remove the first occurence of element item.
     * @param item The item to be removed
     * @return true if item is found and removed; otherwise, return false.
     */
    public boolean remove(E item) {
        if (head == null) {
            return false;
        }
        Node<E> current = head;
        if (item.equals(current.data)) {
            removeFirst();
            return true;
        }
        while (current.next != null) {
            if (item.equals(current.next.data)) {
                removeAfter(current);
                return true;
            }
            current = current.next;
        }
        return false;
    }
```

Chapter	Section	No
2	6	1

Question

For the double-linked list shown in Figure 2.20., assume head references the first list node and tail references the last list node. Write statements to do each of the following.

A. Insert `"Bill"` before `"Tom"`.
B. Insert `"Sue"` before `"Sam"`.
C. Remove `"Bill"`.
D. Remove `"Sam"`.

Solution

```
//Solution to programming exercise 1, Section 6, Chapter 2
```

```
//File: \KW\CH02\KWLinkedList.java startLine: 440 endLine 476
    /** Static method to provide solution to exercise */
    public static void exercise_2_6_1() {
        // Create figure 2.20
        Node tom = new Node<String>("Tom");
        Node dick = new Node<String>("Dick");
        Node harry = new Node<String>("Harry");
        Node sam = new Node<String>("Sam");
        tom.next = dick;
        dick.prev = tom;
        dick.next = harry;
        harry.prev = dick;
        harry.next = sam;
        sam.prev = harry;
        Node<String> head = tom;
        Node<String> tail = sam;
        // Insert "Bill" before "Tom"
        Node<String> bill = new Node<String>("Bill");
        bill.next = tom;
        bill.prev = tom;
        head = tom;
        // Insert "Sue" before "Sam"
        Node sue = new Node("Sue");
        sue.next = sam;
        sue.prev = sam.prev;
        sue.prev.next = sue;
        sam.prev = sue;
        // Remove "Bill"
        head = head.next;
        head.prev = null;
        bill.next = null;
        // Remove "Sam"
        sam.prev.next = sam.next;
        sam.prev = null;
    }
```

Chapter	Section	No
2	7	1

Question

Write the method `indexOf` as specified in the List interface by adapting the code shown in Example 2.14 to return the index of the first occurrence of an object.

Solution

```
//Solution to programming exercise 1, Section 7, Chapter 2
//File: \KW\CH02\KWLinkedList.java startLine: 366 endLine 387
    /**
     * Method to find the index of the first occurence of an item in the list
     * @param target The item being sought
     * @return The index of the first occurence of the tartet item
     *         or -1 if the item is not found.
     */
    @Override
    public int indexOf(Object target) {
        Iterator<E> itr = iterator();
        int index = 0;
        while (itr.hasNext()) {
            if (target.equals(itr.next())) {
                return index;
            } else {
                index++;
            }
        }
        return -1;
    }
```

Chapter	Section	No
2	7	3

Question

Write a method `indexOfMin` that returns the index of the minimum item in a `List`, assuming that each item in the list implements the `Comparable` interface.

Solution

```
//Solution to programming exercise 3, Section 7, Chapter 2
//File: \KW\CH02\KWLinkedList.java startLine: 410 endLine 439
    /**
     * Method to return the index of the minimum item in the list
     * It is assumed that each item in the list implements Comparable
```

```
     * @return Index of the minimum item in the list
     *         or -1 if the list is empty
     * @throws ClassCastExcepition if the list elements do not implement Comparable
     */
    public int indexOfMin() {
        int index = 0;
        int minIndex = 0;
        Iterator<E> itr = iterator();
        Comparable<E> minItem = null;
        if (itr.hasNext()) {
            minItem = (Comparable<E>) itr.next();
        } else {
            return -1;
        }
        while (itr.hasNext()) {
            E nextItem = itr.next();
            index++;
            if (minItem.compareTo(nextItem) >= 0) {
                minItem = (Comparable<E>) nextItem;
                minIndex = index;
            }
        }
        return minIndex;
    }
```

Chapter	Section	No
2	8	1

Question

Implement the `KWListIter.remove` method.

Solution

```
//Solution to programming exercise 1, Section 8, Chapter 2
//File: \KW\CH02\KWLinkedList.java startLine: 309 endLine 349
        /** Remove the last item returned. This can only be
         *  done once per call to next or previous.
         *  @throws IllegalStateException if next or previous
         *  was not called prior to calling this method
         */
        @Override
        public void remove() {
            if (lastItemReturned == null) {
                throw new IllegalStateException();
            }
            // Unlink this item from its next neighbor
            if (lastItemReturned.next != null) {
                lastItemReturned.next.prev = lastItemReturned.prev;
            } else { // Item is the tail
                tail = lastItemReturned.prev;
                if (tail != null) {
                    tail.next = null;
                } else { // list is now empty
                    head = null;
                }
            }
            // Unlink this item from its prev neighbor
            if (lastItemReturned.prev != null) {
                lastItemReturned.prev.next = lastItemReturned.next;
            } else { // Item is the head
                head = lastItemReturned.next;
                if (head != null) {
                    head.prev = null;
                } else {
                    tail = null;
                }
            }
            // Invalidate lastItemReturned
            lastItemReturned = null;
            // decrement both size and index
            size--;
            index--;
        }
```

Chapter	Section	No
2	8	3

Question

Implement the `KWLinkedList` `listIterator` and `iterator` methods.

Solution

```
//Solution to programming exercise 3, Section 8, Chapter 2
//File: \KW\CH02\KWLinkedList.java startLine: 58 endLine 93
    /**
     * Return an Iterator to the list
     * @return an Itertor to the list
     */
    public Iterator<E> iterator() {
        return new KWListIter(0);
    }

    /**
     * Return a ListIterator to the list
     * @return a ListItertor to the list
     */
    public ListIterator<E> listIterator() {
        return new KWListIter(0);
    }

    /** Return a ListIterator that begins at index
     * @param index - The position the iteration is to begin
     * @return a ListIterator that begins at index
     */
    public ListIterator<E> listIterator(int index) {
        return new KWListIter(index);
    }

    /**
     * Return a ListIterator that begins at the same
     * place as an existing ListIterator
     * @param iter - The other ListIterator
     * @return a ListIterator that is a copy of iter
    */
    public ListIterator <E> listIterator(ListIterator <E> iter) {
        return new KWListIter( (KWListIter) iter);
    }
```

Chapter	Section	No
2	9	1

Question

Using either the `KWArrayList` or the `KWLinkedList` as the base, develop an implementation of the `Collection` interface by extending the `AbstractCollection`. Test it by ensuring that the following statements compile:

```
Collection<String> testCollection = new KWArrayList<String>();
Collection<String> testCollection = new KWLinkedList<String>();
```

Solution

```
//Solution to programming exercise 1, Section 9, Chapter 2
//File: \KW\CH02\KWArrayList.java startLine: 12 endLine 15
        extends AbstractList<E>
//Solution to programming exercise 1, Section 9, Chapter 2
//File: \KW\CH02\Exercise_2_9_1.java startLine: 1 endLine 16
// Verification that KWLinkedList and KWArrayList implement the
// Collection interface. This assumes that all other exercises have been
// implemented

import java.util.Collection;

public class Exercise_2_9_1 {

    public static void main(String[] args) {
        Collection<String> textCollection1 = new KWArrayList<String>();
        Collection<String> testCollection2 = new KWLinkedList<String>();
    }
}
```

Chapter	Section	No
2	10	1

Question

Write the code for the other methods of the `OrderedList` class that are listed in Table 2.13.

Solution

```
//Solution to programming exercise 1, Section 10, Chapter 2
//File: \KW\CH02\OrderedList.java startLine: 90 endLine 108
    /**
     * Returns an iterator to this OrderedList.
     * @return The iterator, positioning it before the first element.
```

```
     */
    @Override
    public Iterator<E> iterator() {
        return theList.iterator();
    }

    public int size() {
        return theList.size();
    }

    public void remove(E obj) {
        theList.remove(obj);
    }
```

Chapter	Section	No
2	11	1

Question

Writer a driver program to test method `readInt` using the test data derived for Self-Check Exercise 4, part b.

Solution

```
//Solution to programming exercise 1, Section 11, Chapter 2
//File: \KW\CH02\TestReadInt.java startLine: 0 endLine 115
package KW.CH02;

import java.util.Scanner;

/** Program to test readInt method */
public class TestReadInt {

    private static class TestCase {

        String prompt;
        int minN;
        int maxN;
        int value;
        Class<? extends Throwable> exception;

        public TestCase(String prompt, int minN, int maxN, int value,
                Class<? extends Throwable> exception) {
            this.prompt = prompt;
            this.minN = minN;
            this.maxN = maxN;
            this.value = value;
            this.exception = exception;
        }
    }
    private static TestCase[] testCases = {
        new TestCase(null, 1, 5, 2, null),
        new TestCase("", 1, 5, 6, null),
        new TestCase("MyPrompt", 0, 0, 0, null),
        new TestCase("MyPrompt", 5, 1, 0, IllegalArgumentException.class),
        new TestCase("MyPrompt", -10, 5, 0, null)
    };
    private static Scanner in = new Scanner(System.in);

    /**
     * Method to conduct a test
     * @param prompt The prompt to be displayed
     * @param minN The minimum value allowed
     * @param maxN The maximum value allowed
     * @param value The value to be entered and the expected result
     * @param exception The exception that is expected to be thrown
     * @return true If the test passes
     */
    private static boolean doTest(String prompt, int minN, int maxN,
            int value, Class<? extends Throwable> exception) {
        boolean pass = true;
        if (exception == null) {
            try {
                if (minN <= value && value <= maxN) {
                    System.out.println("The prompt should be " + prompt
                            + "\n Enter the value " + value
                            + "\n Normal return is expected");
                    int result = MyInput.readInt(prompt, minN, maxN);
                    pass = pass && (result == value);
                    System.out.print("Was the prompt properly displayed (y/N)?");
                    String answer = in.next();
                    pass = pass && Character.toUpperCase(answer.charAt(0)) == 'Y';
                    System.out.print("Was the input accepted? (y/N?)");
                    answer = in.next();
```

```
                    pass = pass && Character.toUpperCase(answer.charAt(0)) == 'Y';
                    return pass;
                } else {
                    System.out.println("The prompt should be " + prompt
                            + "\n Enter the value " + value
                            + "\n prompt should be re-displayed"
                            + "\n Enter " + minN + " to the second prompt");
                    int result = MyInput.readInt(prompt, minN, maxN);
                    pass = pass && result == minN;
                    System.out.print("Was the prompt properly displayed (y/N)?");
                    String answer = in.next();
                    pass = pass && Character.toUpperCase(answer.charAt(0)) == 'Y';
                    System.out.print("Was the prompt re-displayed (y/N)?");
                    answer = in.next();
                    pass = pass && Character.toUpperCase(answer.charAt(0)) == 'Y';
                    return pass;
                }
            } catch (Throwable ex) {
                System.err.println("Unexpected exception thrown");
                ex.printStackTrace();
                return false;
            }
        } else {
            try {
                int result = MyInput.readInt(prompt, minN, maxN);
                // Should not get here
                return false;
            } catch (Throwable ex) {
                if (ex.getClass() == exception) {
                    return true;
                } else {
                    System.err.println("Unexpected exception thrown");
                    ex.printStackTrace();
                    return false;
                }
            }
        }
    }

    public static void main(String[] args) {

        for (int i = 0; i < testCases.length; i++) {
            if (doTest(testCases[i].prompt,
                    testCases[i].minN,
                    testCases[i].maxN,
                    testCases[i].value,
                    testCases[i].exception)) {
                System.out.println("Test " + i + " passed");
            } else {
                System.out.println("Test " + i + " failed");
            }
        }
        System.exit(0);
    }
}
```

Chapter	Section	No
2	11	3

Question

Write a `search` method with four parameters: the search array, the target, the start subscript, and the finish subscript. The last two parameters indicate the part of the array that should be searched. Your method should catch or throw exceptions where warranted. Write a driver program to test this method.

Solution

```
//Solution to programming exercise 3, Section 11, Chapter 2
//File: \KW\CH02\SearchArray.java startLine: 0 endLine 111
package KW.CH02;

/**
 * Class to encapsulate searchArray method and its test
 */
public class SearchArray {

    /**
     * Method to search part of an array for a target value
     * @param array The array to search
     * @param target The target value
     * @param minIndex The index to start the search
     * @param maxIndex The index to end the search
     * @return The index of the target or -1 if not found
```

```
     * @throws ArrayIndexOutOfBoundsException if minIndex is &lt; 0
     * or if the target is not found and maxIndex &gt; array.length-1
     * @throws IllegalArgumentException if maxIndex &lt; minIndex
     * @throws NullPointerException if array or target are null
     */
    public static <T> int search(T[] array, T target, int minIndex, int maxIndex) {
        if (maxIndex < minIndex) {
            throw new IllegalArgumentException();
        }
        for (int i = minIndex; i <= maxIndex; i++) {
            if (target.equals(array[i])) {
                return i;
            }
        }
        return -1;
    }

    private static class TestCase {

        private Integer[] array;
        private Integer target;
        private int minIndex;
        private int maxIndex;
        private int result;
        private Class<? extends Throwable> exception;

        public TestCase(Integer[] array,
                Integer target,
                int minIndex,
                int maxIndex,
                int result,
                Class<? extends Throwable> exception) {
            this.array = array;
            this.target = target;
            this.minIndex = minIndex;
            this.maxIndex = maxIndex;
            this.result = result;
            this.exception = exception;
        }
    }

    private static boolean doTest(TestCase t) {
        boolean pass = true;
        try {
            int result = search(t.array, t.target, t.minIndex, t.maxIndex);
            pass = pass && (result == t.result) && (t.exception == null);
        } catch (Throwable ex) {
            if (ex.getClass() == t.exception) {
                return true;
            } else {
                System.err.println("Unexpected Exception");
                ex.printStackTrace();
                return false;
            }
        }
        return pass;
    }
    private static Integer[] array1 = {10, 5, -2, 20, 30, -9};
    private static Integer[] array2 = {99};
    private static Integer[] array3 = new Integer[0];
    private static Integer[] array4 = null;
    private static TestCase[] tests = {
        new TestCase(array1, -2, 0, 5, 2, null),
        new TestCase(array1, -2, 1, 4, 2, null),
        new TestCase(array1, -1, 0, 5, -1, null),
        new TestCase(array1, -1, 1, 4, -1, null),
        new TestCase(array1, -2, 2, 2, 2, null),
        new TestCase(array1, -1, 2, 2, -1, null),
        new TestCase(array1, -1, 2, -1, -1, IllegalArgumentException.class),
        new TestCase(array1, -1, -1, 5, -1, ArrayIndexOutOfBoundsException.class),
        new TestCase(array1, -2, 0, 10, 2, null),
        new TestCase(array1, -1, 0, 10, -1, ArrayIndexOutOfBoundsException.class),
        new TestCase(array2, 99, 0, 0, 0, null),
        new TestCase(array2, -1, 0, 0, -1, null),
        new TestCase(array3, -1, 0, 0, -1, ArrayIndexOutOfBoundsException.class),
        new TestCase(array4, -1, 0, 0, -1, NullPointerException.class),
        new TestCase(array1, null, 0, 0, -1, NullPointerException.class)
    };

    public static void main(String[] args) {
        boolean allPass = true;
        for (int i = 0; i < tests.length; i++) {
            boolean pass = doTest(tests[i]);
```

```
            if (pass) {
                System.out.println("Test " + i + " passed");
            } else {
                System.out.println("Test " + i + "       FAILED");
            }
            allPass = allPass && pass;
        }
        if (allPass) {
            System.out.println("All tests passed");
        }
    }
}
```

Chapter	Section	No
3	1	1

Question

Write a `main` function that creates three stacks of `Integer` objects. Store the numbers -1, 15, 23, 44, 4, 99 in the first two stacks. The top of each stack should store 99.

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 3
//File: \KW\CH03\Exercise_3_1.java startLine: 8 endLine 21
        Stack<Integer> stack1 = new Stack<Integer>();
        Stack<Integer> stack2 = new Stack<Integer>();
        Stack<Integer> stack3 = new Stack<Integer>();

        int[] numbers = {-1, 15, 23, 44, 4, 99};

        for (int i = 0; i < numbers.length; i++) {
            stack1.push(numbers[i]);
            stack2.push(numbers[i]);
        }
```

Chapter	Section	No
3	1	3

Question

Write a second loop to remove a value from the second stack and from the third stack and display each pair of values on a separate output line. Continue until the stacks are empty. Show the output.

Solution

```
//Solution to programming exercise 3, Section 1, Chapter 3
//File: \KW\CH03\Exercise_3_1.java startLine: 28 endLine 41
        while (!stack2.empty()) {
            System.out.println(stack2.pop() + "\t" + stack3.pop());
        }
        /* Expected Output
        99      -1
        4       15
        44      23
        23      44
        15      4
        -1      99
         */
```

Chapter	Section	No
3	2	1

Question

Write a method that reads a line and reverses the words in the line (not the characters) using a stack. For example, given the following input:

```
The quick brown fox jumps over the lazy dog
```

you should get the following output:

```
dog lazy the over jumps fox brown quick The
```

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 3
//File: \KW\CH03\Exercise_3_2.java startLine: 7 endLine 24
    public static String reverseWords(String sentence) {
        String[] words = sentence.split("\\s+");
        Stack<String> stack = new Stack<String>();
        for (String word : words) {
            stack.push(word);
        }
        StringBuilder stb = new StringBuilder();
        while (!stack.empty()) {
```

```
            stb.append(stack.pop());
            if (!stack.empty()) {
                stb.append(" ");
            }
        }
        return stb.toString();
    }
//Solution to programming exercise 1, Section 2, Chapter 3
//File: \KW\CH03\Exercise_3_2_1.java startLine: 7 endLine 24
    public static String reverseWords(String sentence) {
        String[] words = sentence.split("\\s+");
        Stack<String> stack = new Stack<String>();
        for (String word : words) {
            stack.push(word);
        }
        StringBuilder stb = new StringBuilder();
        while (!stack.empty()) {
            stb.append(stack.pop());
            if (!stack.empty()) {
                stb.append(" ");
            }
        }
        return stb.toString();
    }
```

Chapter	Section	No
3	2	3

Question

Code the second approach to finding palindromes.

Solution

```
//Solution to programming exercise 3, Section 2, Chapter 3
//File: \KW\CH03\Exercise_3_2_3.java startLine: 4 endLine 13
    public static boolean isPalindrome(String s) {
        StringBuilder reversed = new StringBuilder();
        for (int i = s.length() - 1; i >= 0; i--) {
            reversed.append(s.charAt(i));
        }
        return s.equalsIgnoreCase(reversed.toString());
    }
```

Chapter	Section	No
3	3	1

Question

Write a method `size` for class `LinkedStack<E>` that returns the number of elements currently on a `LinkedStack<E>`.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 3
//File: \KW\CH03\LinkedStack.java startLine: 102 endLine 117
    /**
     * Method to determine the number of items on the stack
     * @return the number of items on the stack
     */
    public int size() {
        int count = 0;
        Node<E> current = topOfStackRef;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }
```

Chapter	Section	No
3	4	1

Question

Modify class InfixToPostfix to handle the exponentiation operator, indicated by the symbol ^. The first operand is raised to the power indicated by the second operand. Assume that a sequence of ^ operators will not occur and that precedence('^') is greater than precedence('*').

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 3
//File: \KW\CH03\InfixToPostfixParensExp.java startLine: 0 endLine 155
package KW.CH03;
```

```
import java.util.Stack;
import java.util.EmptyStackException;
import java.util.regex.Pattern;
import java.util.Scanner;

/** Translates an infix expression with parentheses
 *  to a postfix expression.
 *  @author Koffman & Wolfgang
 */
public class InfixToPostfixParensExp {

    // Nested Class
    /** Class to report a syntax error. */
    public static class SyntaxErrorException
            extends Exception {

        /**
         * Construct a SyntaxErrorException with the specified
         * message.
         * @param message The message
         */
        SyntaxErrorException(String message) {
            super(message);
        }
    }
    // Data Fields
    /** The operator stack */
    private Stack<Character> operatorStack;
    /** The operators */
    private static final String OPERATORS = "-+*/^()";
    /**
     * The Pattern to extract tokens
     * A token is either a string of digits (\d+)
     * or a JavaIdentifier
     * or an operator
     */
    private static final Pattern pattern =
            Pattern.compile("\\d+\\.\\d*|\\d+|"
            + "\\p{javaJavaIdentifierStart}\\p{javaJavaIdentifierPart}*"
            + "|[" + OPERATORS + "]");
    /** The precedence of the operators, matches order of OPERATORS. */
    private static final int[] PRECEDENCE = {1, 1, 2, 2, 3, -1, -1};
    /** The postfix string */
    private StringBuilder postfix;

    /**
     * Convert a string from infix to postfix.
     * @param infix The infix expression
     * @throws SyntaxErrorException
     */
    public String convert(String infix) throws SyntaxErrorException {
        operatorStack = new Stack<Character>();
        postfix = new StringBuilder();
        Scanner scan = new Scanner(infix);
        try {
            // Process each token in the infix string.
            String nextToken;
            while ((nextToken = scan.findInLine(pattern)) != null) {
                char firstChar = nextToken.charAt(0);
                // Is it an operand?
                if (Character.isJavaIdentifierStart(firstChar)
                        || Character.isDigit(firstChar)) {
                    postfix.append(nextToken);
                    postfix.append(' ');
                } // Is it an operator?
                else if (isOperator(firstChar)) {
                    processOperator(firstChar);
                } else {
                    throw new SyntaxErrorException("Unexpected Character Encountered: "
                            + firstChar);
                }
            } // End while.
            // Pop any remaining operators
            // and append them to postfix.
            while (!operatorStack.empty()) {
                char op = operatorStack.pop();
                // Any '(' on the stack is not matched.
                if (op == '(') {
                    throw new SyntaxErrorException(
                            "Unmatched opening parenthesis");
                }
                postfix.append(op);
```

```
                postfix.append(' ');
            }
            // assert: Stack is empty, return result.
            return postfix.toString();
        } catch (EmptyStackException ex) {
            throw new SyntaxErrorException("Syntax Error: The stack is empty");
        }
    }

    /**
     * Method to process operators.
     * @param op The operator
     * @throws EmptyStackException
     */
    private void processOperator(char op) {
        if (operatorStack.empty() || op == '(') {
            operatorStack.push(op);
        } else {
            // Peek the operator stack and
            // let topOp be the top operator.
            char topOp = operatorStack.peek();
            if (precedence(op) > precedence(topOp)) {
                operatorStack.push(op);
            } else {
                // Pop all stacked operators with equal
                // or higher precedence than op.
                while (!operatorStack.empty()
                        && precedence(op) <= precedence(topOp)) {
                    operatorStack.pop();
                    if (topOp == '(') {
                    }
                    postfix.append(topOp);
                    postfix.append(' ');
                    if (!operatorStack.empty()) {
                        // Reset topOp.
                        topOp = operatorStack.peek();
                    }
                }

                // assert: Operator stack is empty or
                //         current operator precedence >
                //         top of stack operator precedence.
                if (op != ')') {
                    operatorStack.push(op);
                }
            }
        }
    }

    /**
     * Determine whether a character is an operator.
     * @param ch The character to be tested
     * @return true if ch is an operator
     */
    private boolean isOperator(char ch) {
        return OPERATORS.indexOf(ch) != -1;
    }

    /**
     * Determine the precedence of an operator.
     * @param op The operator
     * @return the precedence
     */
    private int precedence(char op) {
        return PRECEDENCE[OPERATORS.indexOf(op)];
    }
}
```

Chapter	Section	No
4	1	1

Question

Write a main function that creates two stacks of `Integer` objects and a queue of `Integer` objects. Store the numbers -1, 15, 23, 44, 4, 99 in the first stack. The top of the stack should store 99.

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 4
//File: \KW\CH04\Exercise_4_1.java startLine: 10 endLine 23
        Stack<Integer> stack1 = new Stack<Integer>();
        Stack<Integer> stack2 = new Stack<Integer>();
        Queue<Integer> queue = new ArrayDeque<Integer>();
```

```
        int[] numbers = {-1, 15, 23, 44, 4, 99};

        for (int i = 0; i < numbers.length; i++) {
            stack1.push(numbers[i]);
            stack2.push(numbers[i]);
        }
```

Chapter	Section	No
4	1	3

Question

Write a second loop to remove a value from the second stack and from the queue and display each pair of values on a separate output line. Continue until the data structures are empty. Show the output.

Solution

```
//Solution to programming exercise 3, Section 1, Chapter 4
//File: \KW\CH04\Exercise_4_1.java startLine: 30 endLine 42
        while (!stack2.empty()) {
            System.out.println(stack2.pop() + "\t" + queue.poll());
        }
        // Expected Output
        // 99      99
        // 4       4
        // 44      44
        // 23      23
        // 15      15
        // -1      -1
```

Chapter	Section	No
4	2	1

Question

Write a `toString` method for class `MaintainQueue`.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 4
//File: \KW\CH04\MaintainQueue.java startLine: 108 endLine 125

    /**
     * Create a string representation of the queue contents with each
     * entry on its own line
     * @return a String representation of the queue contents
     */
    @Override
    public String toString() {
        StringBuilder stb = new StringBuilder();
        for (String next : customers) {
            stb.append(next);
            stb.append("\n");
        }
        return stb.toString();
    }
```

Chapter	Section	No
4	3	1

Question

Write the missing methods required by the `Queue` interface and inner class `Iter` for class `ListQueue<E>`. Class `Iter` should have a data field `current` of type `Node<E>`. Data field `current` should be initialized to `first` when a new `Iter` object is created. Method `next` should return the value of `current` and advance `current`. Method `remove` should throw an `UnsupportedOperationException`.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 4
//File: \KW\CH04\ListQueue.java startLine: 109 endLine 170
    /**
     * Returns the size of the queue
     * @return the size of the queue
     */
    @Override
    public int size() {
        return size;
    }

    /**
     * Returns an iterator to the contents of this queue
```

```
     * @return an iterator to the contents of this queue
     */
    public Iterator<E> iterator() {
        return new Iter();
    }

    /**
     * Inner class to provide an iterator to the contents of
     * the queue.
     */
    private class Iter implements Iterator<E> {

        Node<E> next = front;

        /**
         * Returns true if there are more elements in the iteration
         * @return true if there are more elements in the iteration
         */
        @Override
        public boolean hasNext() {
            return next != null;
        }

        /**
         * Return the next item in the iteration and advance the iterator
         * @return the next item in the iteration
         * @throws NoSuchElementException if the iteration has no more elements
         */
        @Override
        public E next() {
            if (next == null) {
                throw new NoSuchElementException();
            }
            E item = next.data;
            next = next.next;
            return item;
        }

        /**
         * Removes the last item returned by this iteration
         * @throws UnsupportedOperationException this operation is not
         * supported
         */
        @Override
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
```

Chapter Section No
4 3 3

Question

Write the missing methods for class `ArrayQueue<E>` required by the `Queue` interface.

Solution

```
//Solution to programming exercise 3, Section 3, Chapter 4
//File: \KW\CH04\ArrayQueue.java startLine: 108 endLine 127
    /**
     * Return the size of the queue
     * @return The number of items in the queue
     */
    @Override
    public int size() {
        return size;
    }

    /**
     * Returns an iterator to the elements in the queue
     * @return an iterator to the elements in the queue
     */
    @Override
    public Iterator<E> iterator() {
        return new Iter();
    }
```

Chapter Section No
4 3 5

Question

For Self-Check Exercise 3, write methods `offer` and `poll`.

Solution

```
//Solution to programming exercise 5, Section 3, Chapter 4
//File: \KW\CH04\ListQueueReverse.java startLine: 56 endLine 108
    /**
     * Insert an item at the rear of the queue.
     * @post item is added to the rear of the queue.
     * @param item The element to add
     * @return true (always successful)
     */
    @Override
    public boolean offer(E item) {
        // Check for empty queue.
        if (front == null) {
            rear = new Node<E>(item);
            front = rear;
        } else {
            // Allocate a new node at the beginning, store item in it, and
            // link it to old end of queue.
            Node<E> newNode = new Node<E>(item);
            newNode.next = rear;
            rear = newNode;
        }
        size++;
        return true;
    }

    /**
     * Remove the entry at the front of the queue and return it
     * if the queue is not empty.
     * @post front references item that was second in the queue.
     * @return The item removed if successful, or null if not
     */
    @Override
    public E poll() {
        E item = peek(); // Retrieve item at front.
        if (item == null) {
            return null;
        }
        // Remove item at front.
        if (front == rear) {
            front = null;
            rear = null;
        } else {
            Node<E> current = rear;
            while (current.next != front) {
                current = current.next;
            }
            current.next = null;
            front = current;
        }
        size--;
        return item; // Return data at front of queue.
    }
```

Chapter	Section	No
4	4	1

Question

Write a fragment that reads a sequence of strings and inserts each string that is not numeric at the front of a deque and each string that is numeric at the rear of a deque. Your fragment should also count the number of strings of each kind.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 4
//File: \KW\CH04\Exercise_4_4.java startLine: 13 endLine 34
        Pattern isNumeric =
                Pattern.compile("[+-]?(\\d+(\\.\\d*)?([eE][+-]?\\d+)?)|"
                + "(\\.\\d+([eE][+-]?\\d+)?)");
        Deque<String> deque = new ArrayDeque<String>();
        int numNumeric = 0;
        int numOther = 0;
        Scanner in = new Scanner(System.in);
        String token;
        while (in.hasNext()) {
            token = in.next();
            Matcher m = isNumeric.matcher(token);
            if (m.matches()) {
                deque.addLast(token);
                numNumeric++;
            } else {
```

```
                deque.addFirst(token);
                numOther++;
            }
        }
```

Chapter	Section	No
4	4	3

Question

Write a `Deque.addFirst` method for class `ArrayQueue`.

Solution

```
//Solution to programming exercise 3, Section 4, Chapter 4
//File: \KW\CH04\ArrayQueue.java startLine: 65 endLine 76
    public boolean addFirst(E item) {
        if (size == capacity) {
            reallocate();
        }
        size++;
        front = (front - 1 + capacity) % capacity;
        theData[front] = item;
        return true;
    }
```

Chapter	Section	No
4	5	1

Question

Run the AirlineCheckinSim program with a variety of inputs to determine the maximum passenger arrival rate for a given average processing time and the effect of the frequent flyer service policy.

Chapter	Section	No
4	5	3

Question

Write method `enterData` using class `JOptionPane` for data entry.

Solution

```
//Solution to programming exercise 3, Section 5, Chapter 4
//File: \KW\CH04\AirlineSim\AirlineCheckinSim.java startLine: 54 endLine 150
    /** Method to read the simulation parameters */
    private void enterData() {
        String reply;
        reply = JOptionPane.showInputDialog(null,
                "Expected number of frequent flyer arrivals per hour:");
        System.out.println("Expected number of frequent flyer arrivals per hour: "
                + reply);
        frequentFlyerQueue.setArrivalRate(Double.parseDouble(reply) / 60.0);
        reply = JOptionPane.showInputDialog(null,
                "Expected number of regular passenger arrivals per hour:");
        System.out.println("Expected number of regular passenger arrivals per hour: "
                + reply);
        regularPassengerQueue.setArrivalRate(Double.parseDouble(reply) / 60.0);
        reply = JOptionPane.showInputDialog(null,
                "Enter the maximum number of frequent flyers"
                + "\nserved between regualr passengers");
        System.out.println("The maximum number of frequent flyers"
                + "\nserved between regualar passengers: "
                + reply);
        frequentFlyerMax = Integer.parseInt(reply);
        reply = JOptionPane.showInputDialog(null,
                "Enter the maximim service time in minutes");
        System.out.println("Maximum service time in minutes: "
                + reply);
        maxPocessingTime = Integer.parseInt(reply);
        Passenger.setMaxProcessingTime(maxPocessingTime);
        reply = JOptionPane.showInputDialog(null,
                "Enter the total simulation time in minutes");
        System.out.println("The total simulation time in minutes: " + reply);
        totalTime = Integer.parseInt(reply);
        reply = JOptionPane.showInputDialog(null,
```

```
            "Display minute-by-minute trace of simulation (Y or N)");
        System.out.println("Display minute-by-minute trace of simulation (Y or N): "
            + reply);
        showAll = (reply.charAt(0) == 'Y') || (reply.charAt(0) == 'y');
        /*<exercise chapter="4" section="5" type="programming" number="2">*/
        reply = JOptionPane.showInputDialog(null,
            "Set simulation granularity to seconds (Y or N)");
        System.out.println("Set simulation granularity to seconds (Y or N): " + reply);
        if ((reply.charAt(0) == 'Y') || (reply.charAt(0) == 'y')) {
            frequentFlyerQueue.setArrivalRate(frequentFlyerQueue.getArrivalRate() / 60.0);
            regularPassengerQueue.setArrivalRate(regularPassengerQueue.getArrivalRate() / 60.0);
            Passenger.setMaxProcessingTime(Passenger.getMaxProcessingTime() * 60);
            totalTime *= 60;
        }
        /*</exercise>*/
    }

    public void enterData(String fileName) {
        try {
            Properties p = new Properties();
            p.load(new FileReader(new File(fileName)));
            String ffArrivals = p.getProperty("frequentFlyerArrivals");
            if (ffArrivals != null) {
                Queue<int[]> frequentFlyerArrivals = parseArrivalTimes(ffArrivals);
                System.out.println("Frequent Flyer Arrivals \n" + ffArrivals);
                frequentFlyerQueue = new PassengerQueue("Frequent Flyer", frequentFlyerArrivals);
            } else {
                double frequentFlyerArrivalRate =
                    Double.parseDouble(p.getProperty("frequentFlyerArrivalRate"));
                System.out.println("Expected number of frequent flyer arrivals per hour: "
                        + frequentFlyerArrivalRate);
                frequentFlyerQueue.setArrivalRate(frequentFlyerArrivalRate / 60);
            }
            String rpArrivals = p.getProperty("regularPassengerArrivals");
            if (rpArrivals != null) {
                Queue<int[]> regularPassengerArrivals = parseArrivalTimes(rpArrivals);
                System.out.println("Regular Passenger Arrivals \n" + rpArrivals);
                regularPassengerQueue = new PassengerQueue("Regular Passengers",
                                                           regularPassengerArrivals);
            } else {
                double regularPassengerArrivalRate =
                    Double.parseDouble(p.getProperty("regularPassengerArrivalRate"));
                System.out.println("Expected number of regular passenger arrivals per hour: "
                        + regularPassengerArrivalRate);
                regularPassengerQueue.setArrivalRate(regularPassengerArrivalRate / 60);
            }
            if (ffArrivals == null && rpArrivals == null) {
                int maxProcessingTime = Integer.parseInt(p.getProperty("maxProcessingTime"));
                System.out.println("Maximum service time in minutes: "
                        + maxProcessingTime);
                Passenger.setMaxProcessingTime(maxProcessingTime);
            }
            frequentFlyerMax = Integer.parseInt(p.getProperty("frequentFlyerMax"));
            System.out.println("The maximum number of frequent flyers"
                    + "\nserved between regular passengers: "
                    + frequentFlyerMax);
            totalTime = Integer.parseInt(p.getProperty("totalSimulationTime"));
            System.out.println("The total simulation time in minutes: " + totalTime);
            showAll = Boolean.parseBoolean(p.getProperty("showAll"));
            System.out.println("Display minute-by-minute trace of simulation (Y or N): "
                    + showAll);
        } catch (Exception ex) {
            ex.printStackTrace();
            System.exit(1);
        }
    }
```

Chapter	Section	No
5	1	1

Question

Write a recursive method `toNumber` that forms the integer sum of all digit characters in a string. For example, the result of `toNumber("3ac4")` would be 7. Hint: If next is a digit character ('0' through '9'), `Character.isDigit(next)` is true and the numeric value of `next` is `Character.digit(next, 10)`.

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 5
//File: \KW\CH05\RecursiveMethods.java startLine: 237 endLine 250
    public static int toNumber(String s) {
        if (s == null || s.length() == 0) {
            return 0;
        }
        char next = s.charAt(0);
        if (Character.isDigit(next)) {
            return Character.digit(next, 10) + toNumber(s.substring(1));
        } else {
            return toNumber(s.substring(1));
        }
    }
```

Chapter	Section	No
5	1	3

Question

Write a recursive method that implements the recursive algorithm for searching a string in Self-Check Exercise 7. The method heading should be:

```
public static boolean searchString(String str, char ch)
```

Solution

```
//Solution to programming exercise 3, Section 1, Chapter 5
//File: \KW\CH05\RecursiveMethods.java startLine: 261 endLine 272
    public static boolean searchString(String str, char ch) {
        if (str == null || str.length() == 0) {
            return false;
        }
        if (str.charAt(0) == ch) {
            return true;
        }
        return searchString(str.substring(1), ch);
    }
```

Chapter	Section	No
5	2	1

Question

Write a recursive method for raising `x` to the power `n` that works for negative `n` as well as positive `n`. Use the fact that $x^{-n} = \frac{1}{x^n}$.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 5
//File: \KW\CH05\RecursiveMethods.java startLine: 273 endLine 291
    /**
     * Recursive power method (in RecursiveMethods.java).
     * @param x The number being raised to a power
     * @param n The exponent
     * @return x raised to the power n
     */
    public static double power2(double x, int n) {
        if (n < 0) {
            return 1.0 / power2(x, -n);
        }
        if (n == 0) {
            return 1;
        } else {
            return x * power2(x, n - 1);
        }
    }
```

Chapter	Section	No
5	2	3

Question

Modify the `fibonacci` method to throw an illegal argument exception if its argument is less than or equal to zero.

Solution

```
//Solution to programming exercise 3, Section 2, Chapter 5
//File: \KW\CH05\RecursiveMethods.java startLine: 312 endLine 333

    /**
     * Recursive method to calculate Fibonacci numbers
     * (in RecursiveMethods.java).
     * @pre n >= 1
     * @param n The position of the Fibonacci number being calculated
     * @return The Fibonacci number
     * @throws IllegalArgumentException if n <= 0
     */
    public static int fibonacci2(int n) {
        if (n <= 0) {
            throw new IllegalArgumentException("n <= 0: " + n);
        }
        if (n <= 2) {
            return 1;
        } else {
            return fibonacci2(n - 1) + fibonacci2(n - 2);
        }
    }
```

Chapter	Section	No
5	3	1

Question

Write a recursive method to find the sum of all values stored in an array of integers.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 5
//File: \KW\CH05\RecursiveMethods.java startLine: 365 endLine 378
    private static int sum(int[] array, int index) {
        if (index == array.length) {
            return 0;
        } else {
            return array[index] + sum(array, index + 1);
        }
    }

    public static int sum(int[] array) {
        return sum(array, 0);
    }
```

Chapter	Section	No
5	3	3

Question

Implement the method for Self-check Exercise 4. You will need to keep track of the largest value found so far through a method parameter.

Solution

```
//Solution to programming exercise 3, Section 3, Chapter 5
//File: \KW\CH05\RecursiveMethods.java startLine: 379 endLine 403
    /**
     * Method to return the maximum integer in an array
     * @param array The array of integers to search
     * @param index The index of the current value to consider
     * @return the maximum value in the array
     */
    private static int max(int[] array, int index) {
        if (index == array.length - 1) {
            return array[index];
        }
        int maxRest = max(array, index + 1);
        if (array[index] > maxRest) {
            return array[index];
        } else {
            return maxRest;
        }
    }

    /** Wrapper for recursive max method */
    public static int max(int[] array) {
        return max(array, 0);
    }
```

Chapter	Section	No
5	4	1

Question

Write an equals method for the LinkedListRec class that compares this LinkedListRec object to one specified by its argument. Two lists are equal if they have the same number of nodes and store the same information at each node. Don't use the size method.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 5
//File: \KW\CH05\LinkedListRec.java startLine: 176 endLine 223
    /**
     * Method to determine if two LinkedListRec objects are equal.
     * Two LinkeListRec objects are considered equal if they are of the
     * same size and each element in corresponding order are equal using
     * the object's equals method.
     * @param obj The other object
     * @return true of the other obj is equal to this.
     */
    @Override
    @SuppressWarnings("unchecked")
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (this.getClass() != obj.getClass()) {
            return false;
        }
        LinkedListRec<E> other = (LinkedListRec<E>) obj;
        return equals(head, other.head);
    }

    /**
     * Recursive method to determine if two LinkedListRec objects are
     * equal.
     * @param node1 The current node in this list
     * @param node2 The current node in the other list
     */
    private boolean equals(Node<E> node1, Node<E> node2) {
        if (node1 == null & node2 == null) {
            return true;
        }
        if (node1 == null) {
            return false;
        }
        if (node2 == null) {
            return false;
        }
        if (node1.data.equals(node2.data)) {
            return equals(node1.next, node2.next);
        }
        return false;
    }
```

Chapter	Section	No
5	4	3

Question

Write a recursive method `insertBefore` that inserts a specified data object before the first occurrence of another specified data object. For example, the method call `aList.insertBefore(target, inData)` would insert the object referenced by `inData` in a new node just before the first node of `aList` that stores a reference to `target` as its data.

Solution

```
//Solution to programming exercise 3, Section 4, Chapter 5
//File: \KW\CH05\LinkedListRec.java startLine: 252 endLine 292
    /**
     * Method to insert a specified data object before the first
     * occurence of another specified data object. If the object
     * is not found, then the item is inserted at the end of the list.
     * @param target the item that inData is to be inserted before
     * @param inData the item to be inserted
     */
    public void insertBefore(E target, E inData) {
        if (head == null) {
            head = new Node<E>(target, null);
            return;
        }
        if (head.data.equals(target)) {
            head = new Node<E>(inData, head);
            return;
        }
```

```
        insertBefore(target, inData, head);
    }

    /**
     * Recursive method to insert a specified data object before the
     * first occurence of another specified data object. If the object is
     * not found, then the item is inserted at the end of the list.
     * @param target the item that inData is to be inserted before
     * @param inData the item to be inserted
     * @param node the curent node
     */
    private void insertBefore(E target, E inData, Node<E> node) {
        if (node.next == null) {
            node.next = new Node<E>(inData, null);
            return;
        }
        if (target.equals(node.next.data)) {
            node.next = new Node<E>(inData, node.next);
            return;
        }
        insertBefore(target, inData, node.next);
    }
```

Chapter Section No
5 4 5

Question

Code method `insert` in Self-Check Exercise 3.

Solution

```
//Solution to programming exercise 5, Section 4, Chapter 5
//File: \KW\CH05\LinkedListRec.java startLine: 320 endLine 354
    /**
     * Method to insert an object at a specified index
     * @param obj The object to be inserted
     * @param index the index
     */
    public void insert(E obj, int index) {
        if (index < 0) {
            throw new IndexOutOfBoundsException();
        }
        if (index == 0) {
            head = new Node<E>(obj, head);
        } else {
            insert(obj, head, index - 1);
        }
    }

    /**
     * Method to insert an object at a specified index
     * @param obj The object to be inserted
     * @param pred the node preceding the node at the current indes
     * @param index the current index
     */
    private void insert(E obj, Node<E> pred, int index) {
        if (pred == null) {
            throw new IndexOutOfBoundsException();
        }
        if (index == 0) {
            pred.next = new Node<E>(obj, pred.next);
        } else {
            insert(obj, pred.next, index - 1);
        }
    }
```

Chapter Section No
5 5 1

Question

Modify method `countCells`, assuming that cells must have a common side in order to be counted in the same blob. This means that they must be connected horizontally or vertically but not diagonally. Under this condition, the value of the method call aBlob.countCells(4, 1) would be 4 for the grid in Figure 5.16.

Solution

```
//Solution to programming exercise 1, Section 5, Chapter 5
//File: \KW\CH05\Blob.java startLine: 44 endLine 72
    /**
     * Finds the number of cells in the blob at (x,y) assuming
     * that only those cells which are adjacent horizontaly
```

```
     * or vertically are in the blob
     * @pre Abnormal cells are in ABNORMAL color;
     *      Other cells are in BACKGROUND color.
     * @post All cells in the blob are in the TEMPORARY color.
     * @param x The x-coordinate of a blob cell
     * @param y The y-coordinate of a blob cell
     * @return The number of cells in the blob that contains (x, y)
     */
    public int countCells2(int x, int y) {
        int result;

        if (x < 0 || x >= grid.getNCols()
                || y < 0 || y >= grid.getNRows()) {
            return 0;
        } else if (!grid.getColor(x, y).equals(ABNORMAL)) {
            return 0;
        } else {
            grid.recolor(x, y, TEMPORARY);
            return 1
                    + countCells2(x, y + 1) + countCells2(x, y - 1)
                    + countCells2(x - 1, y) + countCells2(x + 1, y);
        }
    }
```

Chapter Section No

5 6 1

Question

Show the interface `GridColors`.

Solution

```
//Solution to programming exercise 1, Section 6, Chapter 5
//File: \KW\CH05\GridColors.java startLine: 1 endLine 18

import java.awt.Color;

/**
 *  An interface for colors
 *@author Koffman and Wolfgang
 */
public interface GridColors {

    Color PATH = Color.green;
    Color BACKGROUND = Color.white;
    Color NON_BACKGROUND = Color.red;
    Color ABNORMAL = NON_BACKGROUND;
    Color TEMPORARY = Color.black;
}
```

Chapter Section No

5 6 3

Question

Write a `Maze.restore` method that restores the maze to its initial state.

Solution

```
//Solution to programming exercise 3, Section 6, Chapter 5
//File: \KW\CH05\Maze.java startLine: 65 endLine 71
    public void restore() {
        resetTemp();
        maze.recolor(PATH, BACKGROUND);
    }
```

Chapter	Section	No
6	3	1

Question

Write a method for the `BinaryTree` class that returns the preorder traversal of a binary tree as a sequence of strings each separated by a space.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 6
//File: \KW\CH06\BinaryTree.java startLine: 183 endLine 207
    /**
     * Method to return the preorder traversal of the binary tree
     * as a sequence of strings each separated by a space.
     * @return A preorder traversal as a string
     */
    public String preorderToString() {
        StringBuilder stb = new StringBuilder();
        preorderToString(stb, root);
        return stb.toString();
    }

    private void preorderToString(StringBuilder stb, Node<E> root) {
        stb.append(root);
        if (root.left != null) {
            stb.append(" ");
            preorderToString(stb, root.left);
        }
        if (root.right != null) {
            stb.append(" ");
            preorderToString(stb, root.right);
        }
    }
```

Chapter	Section	No
6	3	3

Question

Write a method to display the inorder traversal of a binary tree in the same form as Programming Exercise 1, except place a left parenthesis before each subtree and a right parenthesis after each subtree. Don't display anything for an empty subtree. For example the expression tree shown in Figure 6.12 would be represented as
(((x) + (y)) * ((a) / (b))).

Solution

```
//Solution to programming exercise 3, Section 3, Chapter 6
//File: \KW\CH06\BinaryTree.java startLine: 233 endLine 262
    /**
     * A method to display the inorder traversal of a binary tree
     * placeing a left parenthesis before each subtree and a right
     * parenthesis after each subtree. For example the expression
     * tree shown in Figure 6.12 would be represented as
     * (((x) + (y)) * ((a) / (b))).
     * @return An inorder string representation of the tree
     */
    public String inorderToString() {
        StringBuilder stb = new StringBuilder();
        inorderToString(stb, root);
        return stb.toString();
    }

    private void inorderToString(StringBuilder stb, Node<E> root) {
        if (root.left != null) {
            stb.append("(");
            inorderToString(stb, root.left);
            stb.append(") ");
        }
        stb.append(root);
        if (root.right != null) {
            stb.append(" (");
            inorderToString(stb, root.right);
            stb.append(")");
        }
    }
```

Chapter	Section	No
6	4	1

Question

Write methods `contains` and `remove` for the `BinarySearchTree` class. Use methods `find` and `delete` to do the work.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 6
//File: \KW\CH06\BinarySearchTree.java startLine: 177 endLine 199
    /**
     * Removes target from tree.
     * @param target Item to be removed
     * @return true if the object was in the tree, false otherwise
     * @post target is not in the tree
     * @throws ClassCastException if target is not Comparable
     */
    public boolean remove(E target) {
        return delete(target) != null;
    }

    /**
     * Determine if an item is in the tree
     * @param target Item being sought in tree
     * @return true If the item is in the tree, false otherwise
     * @throws ClassCastException if target is not Comparable
     */
    public boolean contains(E target) {
        return find(target) != null;
    }
```

Chapter	Section	No
6	4	3

Question

Write a main method to test a binary search tree based on Listing 2.3. Write a `toString` method that returns the tree contents in ascending order (using an inorder traversal) with newline characters separating the tree elements.

Solution

```
//Solution to programming exercise 3, Section 4, Chapter 6
//File: \KW\CH06\BinarySearchTree.java startLine: 320 endLine 344
    /**
     * Return the contents of the BinarySearchTree as a List of items
     * in ascending order. (Note the exercise suggests returning a string
     * of item separated by newline characters, but a List is more general
     * and useful for the testing performed by the exercise.  Also, the
     * toString method of the BinaryTree class would be hidden and it is
     * useful to verify some of the tests.
     */
    public List<E> toList() {
        List<E> result = new ArrayList<E>();
        toList(result, root);
        return result;
    }

    private void toList(List<E> result, Node<E> node) {
        if (node == null) {
            return;
        }
        toList(result, node.left);
        result.add(node.data);
        toList(result, node.right);
    }
//Solution to programming exercise 3, Section 4, Chapter 6
//File: \KW\CH06\TestBinarySearchTree.java startLine: 0 endLine 109
package KW.CH06;

import java.util.Random;
import java.util.List;

public class TestBinarySearchTree {

    /**
     * Traverses ordered list and displays each element.
     * Displays an error message if an element is out of order.
     * @param testList An ordered list of integers
     * @author Koffman & Wolfgang
     */
    public static void traverseAndShow(BinarySearchTree<Integer> testTree) {
        List<Integer> testList = testTree.toList();
        int prevItem = testList.get(0);

        // Traverse ordered list and display any value that
        // is out of order.
        for (int thisItem : testList) {
            System.out.println(thisItem);
            if (prevItem > thisItem) {
```

```
                System.out.println("*** FAILED, value is "
                        + thisItem);
            }
            prevItem = thisItem;
        }
    }

    public static void main(String[] args) {
        BinarySearchTree<Integer> testTree = new BinarySearchTree<Integer>();
        final int MAX_INT = 500;
        final int START_SIZE = 100;

        // Create a random number generator.
        Random random = new Random();
        for (int i = 0; i < START_SIZE; i++) {
            int anInteger = random.nextInt(MAX_INT);
            testTree.add(anInteger);
        }

        // Add to beginning and end of list.
        testTree.add(-1);
        testTree.add(MAX_INT + 1);
        traverseAndShow(testTree); // Traverse and display.
        List<Integer> testList = testTree.toList();
        // Remove first, last, and middle elements.
        Integer first = testList.get(0);
        Integer last = testList.get(testList.size() - 1);
        Integer middle = testList.get(testList.size() / 2);
        testTree.remove(first);
        testTree.remove(last);
        testTree.remove(middle);
        traverseAndShow(testTree); // Traverse and display.
        // See solution to Exercise 4.2
    }
}
```

Chapter	Section	No
6	5	1

Question

Complete the implementation of the KWPriorityQueue class. Write method swap. Also write methods peek, remove, isEmpty, and size.

Solution

```
//Solution to programming exercise 1, Section 5, Chapter 6
//File: \KW\CH06\KWPriorityQueue.java startLine: 139 endLine 161
    public void swap(int a, int b) {
        E temp = theData.get(a);
        theData.set(a, theData.get(b));
        theData.set(b, temp);
    }

    @Override
    public int size() {
        return theData.size();
    }

    @Override
    public Iterator<E> iterator() {
        return theData.iterator();
    }

    @Override
    public E peek() {
        return theData.get(0);
    }
```

Chapter	Section	No
6	6	1

Question

Write a method `encode` for the `HuffmanTree` class that encodes a `String` of letters that is passed as its first argument. Assume that a second argument, `codes` (type `String[]`), contains the code strings (binary digits) for the symbols (space at position 0, *a* at position 1, *b* at position 2, etc.)

Solution

```
//Solution to programming exercise 1, Section 6, Chapter 6
//File: \KW\CH06\HuffmanTree.java startLine: 152 endLine 209
```

```
/**
 * A method encode for the HuffmanTree class that encodes a String of
 * letters that is passed as its first argument. Assume that a second
 * argument, codes (type String[]), contains the code strings (binary
 * digits) for the symbols (space at position 0, a at position 1, b
 * at position 2, and so on).
 * @param str String to be encoded
 * @param codes Array of codes
 * @param return Encoded string
 * @throws ArrayIndexOutOfBoundsException if str contains a character
 * other than a letter or space.
 */
public static String encode(String str, String[] codes) {
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        int index = 0;
        if (c != ' ') {
            index = Character.toUpperCase(c) - 'A' + 1;
        }
        result.append(codes[index]);
    }
    return result.toString();
}

// The following is needed to run and test the exercise.  Note that it
// uses the Map class which is described in Chapter 7
/**
 * Method to build the code array.  Result is an ordered array of Strings
 * where the first item is the code for the smallest symbol in the
 * array of symbols.
 * @return Array of codes
 */
public String[] getCodes() {
    SortedMap<Character, String> map = new TreeMap<Character, String>();
    String currentCode = "";
    buildCode(map, currentCode, huffTree);
    List<String> codesList = new ArrayList<String>();
    for (Map.Entry<Character, String> e : map.entrySet()) {
        codesList.add(e.getValue());
    }
    return codesList.toArray(new String[codesList.size()]);
}

private void buildCode(Map<Character, String> map,
        String code,
        BinaryTree<HuffData> tree) {
    HuffData theData = tree.getData();
    if (theData.symbol != null) {
        map.put(theData.symbol, code);
    } else {
        buildCode(map, code + "0", tree.getLeftSubtree());
        buildCode(map, code + "1", tree.getRightSubtree());
    }
}
```

Chapter	Section	No
7	1	1

Question

Assume you have declared three sets `a`, `b`, and `c` and that sets `a` and `b` store objects. Write statements that use methods from the `Set` interface to perform the following operations:

a. `c = (a ∪ b)`
b. `c = (a ∩ b)`
c. `c = (a – b)`
d.
```
if (a ⊂ b)
    c = a;
else
    c = b;
```

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 7
//File: \KW\CH07\UseOfSets.java startLine: 38 endLine 65
    public static <E> Set<E> union(Set<E> a, Set<E> b) {
        Set<E> c = new HashSet<E>(a);
        c.addAll(b);
        return c;
    }

    public static <E> Set<E> intersection(Set<E> a, Set<E> b) {
        Set<E> c = new HashSet<E>(a);
        c.retainAll(b);
        return c;
    }

    public static <E> Set<E> difference(Set<E> a, Set<E> b) {
        Set<E> c = new HashSet<E>(a);
        c.removeAll(b);
        return c;
    }

    public static <E> Set<E> setMin(Set<E> a, Set<E> b) {
        if (b.containsAll(a)) {
            return b;
        } else {
            return a;
        }
    }
```

Chapter	Section	No
7	2	1

Question

Write statements to create a `Map` object that will store each word occurring in a term paper along with the number of times the word occurs.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 7
//File: \KW\CH07\WordCount.java startLine: 10 endLine 14
    private static Map<String, Integer> counts =
            new TreeMap<String, Integer>();
```

Chapter	Section	No
7	3	1

Question

Code the following algorithm for finding the location of an object as a static method. Assume a hash table array and an object to be located in the table are passed as arguments. Return the object's position if it is found; return –1 if the object is not found.

1. Compute the index by taking the `hashCode() % table.length`.
2. **`if`** `table[index]` is **`null`**
3. The object is not in the table.
   `else if` `table[index]` is equal to the object
4. The object is in the table.
   `else`
5. Continue to search the table (by incrementing index) until either the object is found or a **`null`** entry is found.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 7
//File: \KW\CH07\HashtableOpen.java startLine: 331 endLine 360
    /**
     * Finds either the target key or the first empty slot in the
     * search chain using linear probing.
```

```
     * @pre The table is not full.
     * @param array The hash table to be searched
     * @param key The key of the target object
     * @return The position of the target or the first empty slot if
     *         the target is not in the table.
     */
    public static int find(Object[] array, Object key) {
        // Calculate the starting index.
        int index = key.hashCode() % array.length;
        if (index < 0) {
            index += array.length; // Make it positive.
        }
        // Increment index until an empty slot is reached
        // or the key is found.
        while ((array[index] != null)
                && (!key.equals(array[index]))) {
            index++;
            // Check for wraparound.
            if (index >= array.length) {
                index = 0; // Wrap around.
            }
        }
        return index;
    }
```

Chapter	Section	No
7	4	1

Question

Write a `remove` method for class `HashtableOpen`.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 7
//File: \KW\CH07\HashtableOpen.java startLine: 235 endLine 254
    /**
     * Remove the item with a given key value
     * @param key The key to be removed
     * @return The value associated with this key, or null
     *         if the key is not in the table.
     */
    @Override
    public V remove(Object key) {
        int index = find(key);
        if (table[index] == null) {
            return null;
        }
        V oldValue = table[index].value;
        table[index] = DELETED;
        numKeys--;
        return oldValue;
    }
```

Chapter	Section	No
7	4	3

Question

Write a `toString` method for class `HashtableOpen`.

Solution

```
//Solution to programming exercise 3, Section 4, Chapter 7
//File: \KW\CH07\HashtableOpen.java startLine: 83 endLine 94
        /**
         * Return a String representation of the Entry
         * @return a String representation of the Entry
         *         in the form key = value
         */
        @Override
        public String toString() {
            return key.toString() + "=" + value.toString();
        }
//Solution to programming exercise 3, Section 4, Chapter 7
//File: \KW\CH07\HashtableOpen.java startLine: 361 endLine 381
    @Override
    public String toString() {
        StringBuilder stb = new StringBuilder();
        stb.append("[");
        boolean first = true;
        for (int i = 0; i < table.length; i++) {
            if (table[i] != null) {
```

```
                stb.append(table[i]);
                if (first) {
                    first = false;
                } else {
                    stb.append(", ");
                }
            }
        }
        stb.append("]");
        return stb.toString();
    }
//Solution to programming exercise 3, Section 4, Chapter 7
//File: \KW\CH07\HashtableChain.java startLine: 69 endLine 81

        /**
         * Return a String representation of the Entry
         * @return a String representation of the Entry
         *         in the form key = value
         */
        @Override
        public String toString() {
            return key.toString() + "=" + value.toString();
        }
```

Chapter Section No
7 4 5

Question

Write a method `size` for both hash table implementations.

Solution

```
//Solution to programming exercise 5, Section 4, Chapter 7
//File: \KW\CH07\HashtableOpen.java startLine: 97 endLine 104
    /** Returns the number of entries in the map */
    @Override
    public int size() {
        return numKeys;
    }
//Solution to programming exercise 5, Section 4, Chapter 7
//File: \KW\CH07\HashtableChain.java startLine: 183 endLine 190
    /** Returns the number of entries in the map */
    @Override
    public int size() {
        return numKeys;
    }
```

Chapter Section No
7 5 1

Question

Write statements to display all key-value pairs in `Map` object `m`, one pair per line. You will need to create an iterator to access the map entries.

Solution

```
//Solution to programming exercise 1, Section 5, Chapter 7
//File: \KW\CH07\Exercise_7_5_1.java startLine: 6 endLine 16
    /**
     *  Method to display the key-value pairs in a Map, one pair per line.
     */
    public static <K, V> void displayMap(Map<K, V> m) {
        for (Map.Entry<K, V> e : m.entrySet()) {
            System.out.println(e.getKey() + "\t" + e.getKey());
        }
    }
```

Chapter Section No
7 5 3

Question

Assume class `HashSetOpen` is written using an array table for storage instead of a `HashMap` object. Write method `contains`.

Solution

```
//Solution to programming exercise 3, Section 5, Chapter 7
//File: \KW\CH07\HashSetOpen2.java startLine: 40 endLine 84
    /**
     * Finds either the target key or the first empty slot in the
     * search chain using linear probing.
     * @pre: The table is not full.
```

```
     * @param key The key of the target object
     * @return The position of the target or the first empty slot if
     *         the target is not in the table.
     */
    private int find(Object key) {
        // Calculate the starting index.
        int index = key.hashCode() % table.length;
        if (index < 0) {
            index += table.length; // Make it positive.
        }
        // Increment index until an empty slot is reached
        // or the key is found.
        while ((table[index] != null)
                && (!key.equals(table[index]))) {
            index++;
            // Check for wraparound.
            if (index >= table.length) {
                index = 0; // Wrap around.
            }
        }
        return index;
    }

   /**
     * Method contains for class HashSetOpen2.
     * @param key The key being sought
     * @return the value associated with this key if found;
     *         otherwise, null
     */
    @Override
    public boolean contains(Object key) {
        // Find the first table element that is empty
        // or the table element that contains the key.
        int index = find(key);

        // If the search is successful, return the value.
return table[index] != null;
    }
```

Chapter	Section	No
7	6	1

Question

Write a class to complete the test of the `MapContactList` class.

Solution

```
//Solution to programming exercise 1, Section 6, Chapter 7
//File: \KW\CH07\MapContactList.java startLine: 73 endLine 91
        //Display should be
        //Jones
        //[123, 345]
        //King
        //[135, 357]
        contactList.display();
        // Jones should be in the list
        System.out.println("Numbers for Jones "
                + contactList.lookupEntry("Jones"));
        // Smith should not
        System.out.println("Numbers for Smith "
                + contactList.lookupEntry("Smith"));
        contactList.removeEntry("Jones");
        // Jones should no longer be in the list
        System.out.println("Numbers for Jones "
                + contactList.lookupEntry("Jones"));
```

Chapter	Section	No
7	7	1

Question

Write a program fragment to display the elements of a set s in normal order and then in reverse order.

Solution

```
//Solution to programming exercise 1, Section 7, Chapter 7
//File: \KW\CH07\Exercise_7_7_1.java startLine: 15 endLine 27
        // Display contents of s in normal order
        System.out.println("Normal Order");
        for (Integer i : s) {
            System.out.println(i);
        }
```

```
        // Dispaly contents of s in reverse order
        System.out.println("Reverse Order");
        for (Integer i : s.descendingSet()) {
            System.out.println(i);
        }
```

Chapter	Section	No
7	7	3

Question

Code method `computeGaps` from Self-Check Exercise 2.

Solution

```
//Solution to programming exercise 3, Section 7, Chapter 7
//File: \KW\CH07\NavigableMapUtils.java startLine: 95 endLine 204
    /**
     * A SkipMap is a non-modifiable map view of a map that
     * starts at the  start item, and contains every gap items
     * in the original map.
     */
    private static class SkipMap<K, V> extends AbstractMap<K, V> {

        Set<Map.Entry<K, V>> entrySet;

        /**
         * Construct a new SkipMap
         * @param map The original map
         * @param start The index of the first item
         * @param gap The number of items between items
         */
        public SkipMap(Map<K, V> map, int start, int gap) {
            entrySet = new SkipSet(map.entrySet(), start, gap);
        }

        /**
         * Return a Set view of the mappings contained in this map
         * @return a Set view of the mappings contained in this map
         */
        public Set<Map.Entry<K, V>> entrySet() {
            return entrySet;
        }
    }

    /**
     * A SkipSet is a Set view of a set that starts at the
     * start item and contains every gap items
     */
    private static class SkipSet<E> extends AbstractSet<E> {

        private Set<E> theSet;
        private int start;
        private int gap;
        private int size;

        /**
         * Construct a new SkipSet
         * @param theSet The set
         * @param start The start index
         * @param gap The gap
         */
        public SkipSet(Set<E> theSet, int start, int gap) {
            this.theSet = theSet;
            this.start = start;
            this.gap = gap;
            size = (theSet.size() - start) / gap;
        }

        /**
         * Return the size of this set
         */
        public int size() {
            return size;
        }

        /**
         * Skip iterator
         */
        private class SkipIterator implements Iterator<E> {

            private int index = 0;
            private int pos = 0;
```

```
            private Iterator<E> baseIterator;

            public SkipIterator() {
                index = start;
                baseIterator = theSet.iterator();
            }

            public boolean hasNext() {
                return index < theSet.size();
            }

            public E next() {
                while (pos < index) {
                    baseIterator.next();
                    pos++;
                }
                pos++;
                index += gap;
                return baseIterator.next();
            }

            public void remove() {
                throw new UnsupportedOperationException();
            }
        }

        public Iterator<E> iterator() {
            return new SkipIterator();
        }
    }

    public static List<Double> computeGaps(Map<Integer, Double> valueMap,
            int gap) {
        List<Double> result = new ArrayList<Double>();
        for (int i = 0; i < gap; i++) {
            double average =
                    computeAverage(new SkipMap<Integer, Double>(valueMap, i, gap));
            result.add(average);
        }
        return result;
    }
```

Chapter	Section	No
8	1	1

Question

Write a method call to sort the last half of array people using the natural ordering.

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 8
//File: \KW\CH08\SortPeople.java startLine: 8 endLine 19
    /**
     * Method to sort the last half of an array of Person objects
     * @param people The array of Person objects to be sorted
     * @post the second half of the array is sorted by the natural
     * ordering of the Person class
     */
    public static void sortSecondHalfNatural(Person[] people) {
        Arrays.sort(people, people.length / 2, people.length);
    }
```

Chapter	Section	No
8	1	3

Question

Write a method call to sort peopleList using the natural ordering.

Solution

```
//Solution to programming exercise 3, Section 1, Chapter 8
//File: \KW\CH08\SortPeople.java startLine: 33 endLine 44
    /**
     * Method to sort a List of Person objects
     * @param peopleList The list of Person objects to be sorted
     * @post the second half of the array is sorted by the natural
     * ordering of the Person class
     */
    public static void sortSecondHalf(List<Person> peopleList) {
        Collections.sort(peopleList);
    }
```

Chapter	Section	No
8	2	1

Question

Modify the selection sort method to sort the elements in decreasing order and to incorporate the change in Self-Check Exercise 3.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 8
//File: \KW\CH08\ReverseSelectionSort.java startLine: 0 endLine 42
package KW.CH08;

/** Implements the selection sort algorithm.
 *  @author Koffman and Wolfgang
 **/
public class ReverseSelectionSort implements SortAlgorithm {

    /**
     * Sort the array in reverse order using the Selection Sort algorithm.
     * @pre  table contains Comparable objects.
     * @post table is sorted.
     * @param table The array to be sorted
     */
    @Override
    public <T extends Comparable<T>>  void sort(T[] table) {
        int n = table.length;
        for (int fill = 0; fill < n - 1; fill++) {
            // Invariant: table[0 . . . fill - 1] is sorted.
            int posMax = fill;
            for (int next = fill + 1; next < n; next++) {
                // Invariant: table[posMax] is the largest item in
                // table[fill . . . next - 1].
                if (table[next].compareTo(table[posMax]) > 0) {
                    posMax = next;
                }
            }
            // assert: table[posMax] is the largest item in
            // table[fill . . . n - 1].
            // Exchange table[fill] and table[posMax].
```

```
            if (fill != posMax) {
                T temp = table[fill];
                table[fill] = table[posMax];
                table[posMax] = temp;
            }
            // assert: table[fill] is the largest item in
            // table[fill . . . n - 1].
        }
        // assert: table[0 . . . n - 1] is sorted in reverse order
    }
}
```

Chapter	Section	No
8	3	1

Question

Add statements to trace the progress of bubble sort. Display the array contents after each pass is completed.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 8
//File: \KW\CH08\BubbleSortWithTrace.java startLine: 0 endLine 65
package KW.CH08;

import java.util.*;

public class BubbleSortWithTrace {

    public static <T extends Comparable<T>>  void sort(T[] table) {
        CountingComparator<T> c = new CountingComparator<T>();
        int pass = 0;
        c.clear();

        int numExchanges = 0;
        boolean exchanges = false;
        System.out.print(pass + "\t");
        printArray(table);
        System.out.println("\t" + c.getCount() + "\t" + numExchanges);
        pass++;

        do {
            exchanges = false;
            numExchanges = 0;
            c.clear();

            for (int i = 0; i < (table.length - pass); i++) {
                if (c.compare(table[i], table[i + 1]) > 0) {
                    ++numExchanges;

                    T temp = table[i];
                    table[i] = table[i + 1];
                    table[i + 1] = temp;
                    exchanges = true;
                }
            }

            System.out.print(pass + "\t");
            printArray(table);
            System.out.println("\t" + c.getCount() + "\t"
                    + numExchanges);
            pass++;
        } while (exchanges);
    }

    private static <T> void printArray(T[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i]);

            if (i < (array.length + 1)) {
                System.out.print(" ");
            }
        }
    }

    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();

        for (String s : args) {
            list.add(new Integer(s));
        }
        Integer[] array = list.toArray(new Integer[list.size()]);
        sort(array);
```

```
    }
}
```

Chapter	Section	No
8	4	1

Question

Eliminate method insert in Listing 8.3 and write its code inside the for statement.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 8
//File: \KW\CH08\InsertionSortMod.java startLine: 6 endLine 32

    /**
     * Sort the table using insertion sort algorithm.
     * @pre  table contains Comparable objects.
     * @post table is sorted.
     * @param table The array to be sorted
     */
    @Override
    public <T extends Comparable<T>>  void sort(T[] table) {
        for (int nextPos = 1; nextPos < table.length; nextPos++) {
            // Invariant: table[0 . . . nextPos - 1] is sorted.
            // Insert element at position nextPos
            // in the sorted subarray.
            T nextVal = table[nextPos]; // Element to insert.
            int nextInsert = nextPos;
            while (nextInsert > 0
                    && nextVal.compareTo(table[nextInsert - 1]) < 0) {
                table[nextInsert] = table[nextInsert - 1]; // Shift down.
                nextInsert--; // Check next smaller element.
            }
            // Insert nextVal at nextPos.
            table[nextInsert] = nextVal;
        } // End for.
    } // End sort.
```

Chapter	Section	No
8	6	1

Question

Eliminate method insert in Listing 8.4 and write its code inside the for statement.

Solution

```
//Solution to programming exercise 1, Section 6, Chapter 8
//File: \KW\CH08\ShellSortMod.java startLine: 7 endLine 43

    /**
     * Sort the table using Shell sort algorithm.
     * @pre  table contains Comparable objects.
     * @post table is sorted.
     * @param table The array to be sorted
     */
    @Override
    public <T extends Comparable<T>>  void sort(T[] table) {
        // Gap between adjacent elements.
        int gap = table.length / 2;
        while (gap > 0) {
            for (int nextPos = gap; nextPos < table.length;
                    nextPos++) {
                // Insert element at nextPos in its subarray.
                int nextInsert = nextPos;
                T nextVal = table[nextInsert]; // Element to insert.
                // Shift all values > nextVal in subarray down by gap.
                while ((nextInsert > gap - 1) // First element not shifted.
                        && (nextVal.compareTo(table[nextInsert - gap]) < 0)) {
                    table[nextInsert] = table[nextInsert - gap]; // Shift down.
                    nextInsert -= gap; // Check next position in subarray.
                }
                table[nextInsert] = nextVal; // Insert nextVal.
            } // End for.

            // Reset gap for next pass.
            if (gap == 2) {
                gap = 1;
            } else {
                gap = (int) (gap / 2.2);
            }
        } // End while.
    } // End sort.
```

Chapter	Section	No
8	7	1

Question

Add statements that trace the progress of method sort by displaying the array table after each merge operation. Also display the arrays referenced by `leftTable` and `rightTable`.

Solution

```
//Solution to programming exercise 1, Section 7, Chapter 8
//File: \KW\CH08\MergeSortWithTrace.java startLine: 0 endLine 81
package KW.CH08;

import java.util.*;

public class MergeSortWithTrace {

    public static void sort(int[] a, int level) {
        for (int kk = 0; kk < level; kk++) {
            System.out.print("\t");
        }
        System.out.printf("sort(%s)%n", arrayToString(a));
        if (a.length > 1) {
            int halfSize = a.length / 2;
            for (int kk = 0; kk < level + 1; kk++) {
                System.out.print("\t");
            }
            System.out.printf("halfSize = %d%n", halfSize);
            int[] leftTable = new int[halfSize];
            int[] rightTable = new int[a.length - halfSize];
            System.arraycopy(a, 0, leftTable, 0, halfSize);
            System.arraycopy(a, halfSize, rightTable, 0, a.length - halfSize);
            for (int kk = 0; kk < level + 1; kk++) {
                System.out.print("\t");
            }
            System.out.printf("leftTable = %s%n", arrayToString(leftTable));
            for (int kk = 0; kk < level + 1; kk++) {
                System.out.print("\t");
            }
            System.out.printf("rightTable = %s%n", arrayToString(rightTable));
            sort(leftTable, level + 1);
            sort(rightTable, level + 1);
            merge(leftTable, rightTable, a);
            for (int kk = 0; kk < level + 1; kk++) {
                System.out.print("\t");
            }
            System.out.printf("merge(%s, %s) = %s%n", arrayToString(leftTable),
                    arrayToString(rightTable), arrayToString(a));
        }
    }

    public static void merge(int[] left, int[] right, int[] a) {
        int i = 0;
        int j = 0;
        int k = 0;
        while (i < left.length && j < right.length) {
            if (left[i] < right[j]) {
                a[k++] = left[i++];
            } else {
                a[k++] = right[j++];
            }
        }
        while (i < left.length) {
            a[k++] = left[i++];
        }
        while (j < right.length) {
            a[k++] = right[j++];
        }
    }

    public static String arrayToString(int[] a) {
        ArrayList<Integer> l = new ArrayList<Integer>();
        for (int x : a) {
            l.add(x);
        }
        return l.toString();
    }

    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();
        for (String s : args) {
```

```
            list.add(new Integer(s));
        }
        int[] array = new int[list.size()];
        int[] array = new int[list.size()];
        for (int i = 0; i < list.size(); i++) {
            array[i] = list.get(i);
        }
        sort(array, 0);
    }
}
```

Chapter	Section	No
8	9	1

Question

Insert statements to trace the quicksort algorithm. After each call to partition, display the values of first, pivIndex, and last and the array.

Solution

```
//Solution to programming exercise 1, Section 9, Chapter 8
//File: \KW\CH08\QuickSort1WithTrace.java startLine: 0 endLine 89
package KW.CH08;

import java.util.*;

public class QuickSort1WithTrace {

    public static <T extends Comparable<T>>  void sort(T[] table) {
        sort(table, 0, table.length - 1, 0);
    }

    private static <T extends Comparable<T>>  void sort(T[] table,
            int first,
            int last,
            int level) {
        for (int i = 0; i < level; i++) {
            System.out.print(" ");
        }
        System.out.printf("sort(%s, %d, %d)%n",
                arrayToString(table), first, last);
        if (first < last) {
            for (int i = 0; i < level; i++) {
                System.out.print(" ");
            }
            int pivIndex = partition(table, first, last);
            System.out.println("pivIndex: " + pivIndex);
            sort(table, first, pivIndex - 1, level + 1);
            for (int i = 0; i < level + 1; i++) {
                System.out.print(" ");
            }
            System.out.printf("table: %s%n", arrayToString(table));
            sort(table, pivIndex + 1, last, level + 1);
            for (int i = 0; i < level + 1; i++) {
                System.out.print(" ");
            }
            System.out.printf("table: %s%n", arrayToString(table));
        }
    }

    private static <T extends Comparable<T>>  int partition(T[] table,
            int first,
            int last) {
        T pivot = table[first];
        int up = first;
        int down = last;
        do {
            while ((up < last) && (pivot.compareTo(table[up]) >= 0)) {
                ++up;
            }
            while (pivot.compareTo(table[down]) < 0) {
                --down;
            }
            if (up < down) {
                swap(table, up, down);
            }
        } while (up < down);
        swap(table, first, down);
        return down;
    }
    private static <T> void swap(T[] table, int i, int j) {
        T temp = table[i];
        table[i] = table[j];
```

```
        table[j] = temp;
    }

    private static <T> String arrayToString(T[] array) {
        StringBuilder stb = new StringBuilder("[");
        for (int i = 0; i < array.length; i++) {
            stb.append(array[i]);
            if (i < array.length - 1) {
                stb.append(", ");
            } else {
                stb.append("]");
            }
        }
        return stb.toString();
    }

    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();
        for (String s : args) {
            list.add(new Integer(s));
        }
        Integer[] array = list.toArray(new Integer[list.size()]);
        sort(array);
    }
}
```

Chapter	Section	No
8	10	1

Question

Write method dumpTable.

Solution

```
//Solution to programming exercise 1, Section 10, Chapter 8
//File: \KW\CH08\TestSort.java startLine: 103 endLine 123
    private static void dumpTable(Integer[] thetable) {
        if (thetable.length <= 20) {
            for (int i = 0; i < thetable.length; i++) {
                System.out.println(i + ": " + thetable[i]);
            }
        } else {
            int mid = 10;
            for (int i = 0; i < mid; i++) {
                System.out.println(i + ": " + thetable[i]);
            }
            if (mid == 10) {
                System.out.println(". . .");
            }
            for (int i = thetable.length - mid; i < thetable.length; i++) {
                System.out.println(i + ": " + thetable[i]);
            }
        }
    }
```

Chapter	Section	No
8	10	3

Question

Extend the driver to test all O(n log n) sorts and collect statistics on the different sorting algorithms. Test the sorts using an array of random numbers and also a data file processed by the solution to Programming Exercise 2.

Solution

```
//Solution to programming exercise 3, Section 10, Chapter 8
//File: \KW\CH08\TestSorts.java startLine: 0 endLine 161
package KW.CH08;

import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Random;
import java.util.Scanner;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.ArrayList;

/**
 * Program to test and time SortAlgorithms
 * @author Paul Wolfgang
```

```
 */
public class TestSorts {

    /** Array of SortAlgorithms */
    private static SortAlgorithm[] algorithms = {new SelectionSort(),
        new BubbleSort(),
        new InsertionSort(),
        new InsertionSortMod(),
        new ShellSort(),
        new ShellSortMod(),
        new MergeSort(),
        new HeapSort(),
        new QuickSort1(),
        new QuickSort2(),
        new QuickSort3()
    };
    /** Array of sorting times */
    double[] times = new double[algorithms.length];

    /**
     * Method to time a sort algorithm.  The algorithm is run 11 times.
     * The result of the first time is not kept and the average of the
     * other 10 is computed.
     * @param algorithm The algorithm to time
     * @param table The table of values to sort
     * @return average runtime in seconds
     */
    private static double timeSort(SortAlgorithm algorithm, Integer[] table) {
        System.out.println("Sorting using " + algorithm.getClass().getName());
        Integer[] copy = new Integer[table.length];
        System.arraycopy(table, 0, copy, 0, table.length);
        algorithm.sort(table);
        long totalTime = 0;
        for (int i = 0; i < 10; i++) {
            System.arraycopy(copy, 0, table, 0, table.length);
            long startTime = System.nanoTime();
            algorithm.sort(table);
            long endTime = System.nanoTime();
            totalTime += (endTime - startTime);
        }
        return (double) (totalTime) / 10e9;
    }

    /**
     * Method to time the Arrays.sort algorithm.  The algorithm is run
     * 11 times. The result of the first time is not kept and the average
     * of the other 10 is computed.
     * @param algorithm The algorithm to time
     * @param table The table of values to sort
     * @return average runtime in seconds
     */
    private static double timeArraysSort(Integer[] table) {
        Integer[] copy = new Integer[table.length];
        System.arraycopy(table, 0, copy, 0, table.length);
        Arrays.sort(table);
        long totalTime = 0;
        for (int i = 0; i < 10; i++) {
            System.arraycopy(copy, 0, table, 0, table.length);
            long startTime = System.nanoTime();
            Arrays.sort(table);
            long endTime = System.nanoTime();
            totalTime += (endTime - startTime);
        }
        return (double) (totalTime) / 10e9;
    }

    /**
     * Verifies that the elements in array are in increasing order
     * @param test The array to verify
     * @return true if the elements are in increasing order
     */
    private static <T extends Comparable<T>>  boolean verify(T[] test) {
        boolean ok = true;
        for (int i = 1; ok && i < test.length; i++) {
            ok = test[i - 1].compareTo(test[i]) <= 0;
        }
        return ok;
    }

    /**
     * Main method to time and test sort algorithms.  A table of random
     * integers is generated and sorted using all of the sort algorithms.
     * Timing data is collected and compared to the timing for Arrays.sort.
```

```
     * @param args[0] If present is the size of array to test if it is an
     * integer.  It not an integer, it is then assumed to be the name of
     * a file that contains the data to be sorted
     */
    public static void main(String[] args) {
        int size = -1;
        Integer[] original = null;
        if (args.length > 0) {
            try {
                size = Integer.parseInt(args[0]);
            } catch (NumberFormatException ex) {
                try {
                    Scanner in = new Scanner(new FileReader(args[0]));
                    List<Integer> list = new ArrayList<Integer>();
                    while (in.hasNextInt()) {
                        list.add(in.nextInt());
                    }
                    size = list.size();
                    original = list.toArray(new Integer[size]);
                } catch (IOException ioex) {
                    System.err.println("Error reading from " + args[0]);
                    System.err.println(ioex);
                }
            }
        }
        if (size < 0) {
            Scanner scan = new Scanner(System.in);
            while (size < 0) {
                System.out.print("Enter the size of the array: ");
                try {
                    size = scan.nextInt();
                } catch (InputMismatchException ex) {
                    // Ignore this exception
                }
            }
        }
        //size >= 0
        if (original == null) {
            Random rand = new Random();
            original = new Integer[size];
            for (int i = 0; i < size; i++) {
                original[i] = rand.nextInt();
            }
        }
        double[] times = new double[algorithms.length];
        for (int j = 0; j < algorithms.length; j++) {
            Integer[] table = Arrays.copyOf(original, original.length);
            times[j] = timeSort(algorithms[j], table);
            if (!verify(table)) {
                System.err.println(algorithms[j].getClass() + " failed");
            }
        }
        double arraysSortTime = timeArraysSort(original);
        System.out.printf("Arrays.sort took %.3f seconds to sort %d integers%n",
                arraysSortTime, size);
        for (int j = 0; j < algorithms.length; j++) {
            System.out.printf("%s took %.3f seconds%n",
                    algorithms[j].getClass().getName(), times[j]);
        }
    }
}
```

Chapter	Section	No
8	11	1

Question

Adapt the Dutch National Flag algorithm to do the quicksort partitioning. Consider the red region to be those values less than the pivot, the white region to be those values equal to the pivot, and the blue region to be those values equal to the pivot. You should initially sort the first, middle, and last items and use the middle value as the pivot value.

Solution

```
//Solution to programming exercise 1, Section 11, Chapter 8
//File: \KW\CH08\QuickSort3.java startLine: 9 endLine 74
    /**
     * Sort a part of the table using the quicksort algorithm.
     * @ @post The part of table from first through last is sorted.
     * @param table The array to be sorted
     * @param first The index of the low bound
     * @param last The index of the high bound
     */
```

```
    @Override
    protected <T extends Comparable<T>>  void quickSort(T[] table,
            int first,
            int last) {
        if (first < last) { // There is data to be sorted.
            // Partition the table.
            int[] pivIndex = partition3(table, first, last);
            // Sort the left half.
            quickSort(table, first, pivIndex[0] - 1);
            // Sort the right half.
            quickSort(table, pivIndex[1] + 1, last);
        }
    }

    /**
     * Partition the table so that values from first to pivIndex[0]
     * are less than the pivot value, and values from
     * pivIndex[1] to last are greater than the pivot value.
     * This partitioning algorithm is based upon Dijkstra's
     * Dutch National Flag problem.
     * @param table The table to be partitioned
     * @param first The index of the low bound
     * @param last  The index of the high bound
     * @return The boundaries of the pivot value
     */
    protected <T extends Comparable<T>>  int[] partition3(T[] table,
            int first,
            int last) {
        // Put the median of table[first], table[middle], table[last]
        // int table[first] and use this value as the pivot
        bubbleSort3(table, first, last);
        // Pick the middle value for the pivot.
        T pivot = table[first + (last - first) / 2];
        int less = first;
        int equal = last;
        int greater = last;
        // Invariant:
        // for all i, 0 <= i <=last
        // 0 <= i < less ==> table[i] < pivot
        // less <= i <= equal table[i] is unknown
        // equal < i < greater table[i] == pivot
        // greater < i <= last table[i] > pivot
        while (less <= equal) {
            if (table[equal].compareTo(pivot) == 0) {
                equal--;
            } else if (table[equal].compareTo(pivot) < 0) {
                swap(table, less, equal);
                less++;
            } else {
                swap(table, equal, greater);
                equal--;
                greater--;
            }
        }
        return new int[]{less, greater};
    }
```

Chapter	Section	No
9	1	1

Question

Add the `rotateLeft` method to the `BinarySearchTreeWithRotate` class.

Solution

```
//Solution to programming exercise 1, Section 1, Chapter 9
//File: \KW\CH09\BinarySearchTreeWithRotate.java startLine: 33 endLine 52
    /**
     * Method to perform a left rotation (rotateLeft).
     * @pre  localRoot is the root of a binary search tree
     * @post localRoot.right is the root of a binary search tree
     *       localRoot.right.right is raised one level
     *       localRoot.right.left does not change levels
     *       localRoot.left is lowered one level
     *       the new localRoot is returned.
     * @param localRoot The root of the binary tree to be rotated
     * @return the new root of the rotated tree
     */
    protected Node<E> rotateLeft(Node<E> localRoot) {
        Node<E> temp = localRoot.right;
        localRoot.right = temp.left;
        temp.left = localRoot;
        return temp;
    }
```

Chapter	Section	No
9	2	1

Question

Program the `rebalanceRight` method.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 9
//File: \KW\CH09\AVLTree.java startLine: 168 endLine 212
    /**
     * Method to rebalance right.
     * @pre localRoot is the root of an AVL subtree that is
     *      critically right-heavy.
     * @post Balance is restored.
     * @param localRoot Root of the AVL subtree
     *        that needs rebalancing
     * @return a new localRoot
     */
    private AVLNode<E> rebalanceRight(AVLNode<E> localRoot) {
        // Obtain reference to right child.
        AVLNode<E> rightChild = (AVLNode<E>) localRoot.right;
        // See whether right-left heavy.
        if (rightChild.balance < AVLNode.BALANCED) {
            // Obtain reference to right-left child.
            AVLNode<E> rightLeftChild = (AVLNode<E>) rightChild.left;
            // Adjust the balances to be their new values after
            // the rotations are performed.
            if (rightLeftChild.balance > AVLNode.BALANCED) {
                rightChild.balance = AVLNode.RIGHT_HEAVY;
                rightLeftChild.balance = AVLNode.BALANCED;
                localRoot.balance = AVLNode.BALANCED;
            } else if (rightLeftChild.balance < AVLNode.BALANCED) {
                rightChild.balance = AVLNode.BALANCED;
                rightLeftChild.balance = AVLNode.BALANCED;
                localRoot.balance = AVLNode.LEFT_HEAVY;
            } else {
                rightChild.balance = AVLNode.BALANCED;
                localRoot.balance = AVLNode.BALANCED;
            }
            // Perform right rotation.
            localRoot.right = rotateRight(rightChild);
        } else { //right-right case
            // In this case the rightChild (the new root)
            // and the root (new left child) will both be balanced
            // after the rotation.
            rightChild.balance = AVLNode.BALANCED;
            localRoot.balance = AVLNode.BALANCED;
        }
        // Now rotate the local root left.
        return (AVLNode<E>) rotateLeft(localRoot);
    }
```

Chapter	Section	No
9	2	3

Question

Program the `incrementBalance` method.

Solution

```
//Solution to programming exercise 3, Section 2, Chapter 9
//File: \KW\CH09\AVLTree.java startLine: 233 endLine 253
    /**
     * Method to increment the balance field and to reset the value of
     * increase.
     * @pre The balance field was correct prior to an insertion [or
     *      removal,] and an item is either been added to the right[ or removed
     *      from the left].
     * @post The balance is incremented and the increase flags is set to
     *       false if the overall height of this subtree has not changed.
     * @param node The AVL node whose balance is to be incremented
     */
    private void incrementBalance(AVLNode<E> node) {
        // Decrement the balance.
        node.balance++;
        if (node.balance == AVLNode.BALANCED) {
            // If now balanced, overall height has not increased.
            increase = false;
        }
    }
```

Chapter	Section	No
9	3	1

Question

Program the case where the item is greater than root.data.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 9
//File: \KW\CH09\RedBlackTree.java startLine: 125 endLine 157
            if (localRoot.right == null) {
                // create new right child
                localRoot.right = new RedBlackNode<E>(item);
                addReturn = true;
                return localRoot;
            } else { // need to search
                // check for two red children swap colors
                moveBlackDown(localRoot);
                // recursively insert on the right
                localRoot.right =
                        add((RedBlackNode<E>) localRoot.right, item);
                // see if the right child is now red
                if (((RedBlackNode) localRoot.right).isRed) {
                    if (localRoot.right.right != null && ((RedBlackNode) localRoot.right.right).isRed)
{
                        // right-right grandchild is also red
                        // single rotate is necessary
                        ((RedBlackNode) localRoot.right).isRed = false;
                        localRoot.isRed = true;
                        return rotateLeft(localRoot);
                    } else if (localRoot.right.left != null && ((RedBlackNode<E>)
localRoot.right.left).isRed) {
                        // left-right grandchild is also red
                        // double rotate is necessary
                        localRoot.right = rotateRight(localRoot.right);
                        ((RedBlackNode<E>) localRoot.right).isRed = false;
                        localRoot.isRed = true;
                        return rotateLeft(localRoot);
                    }
                }
                return localRoot;
            }
```

Chapter	Section	No
9	5	1

Question

Code the `binarySearch` method of the `BTree`.

Solution

```
//Solution to programming exercise 1, Section 5, Chapter 9
//File: \KW\CH09\BTree.java startLine: 261 endLine 293
```

```
    /**
     * Perform a binary search of the array data for target.
     * @param target The item being sought
     * @param data The sorted array the may contain the data
     * @param first The first index to be searched
     * @param last One past the last index to be searched
     * @return The smallest index such that target >= data[index]
     */
    private int binarySearch(E target, E[] data, int first, int last) {
        if (first == last) {
            return first;
        }
        if (last - first == 1) {
            if (target.compareTo(data[first]) <= 0) {
                return first;
            } else {
                return last;
            }
        }
        int middle = first + (last - first) / 2;
        int compResult = target.compareTo(data[middle]);
        if (compResult == 0) {
            return middle;
        }
        if (compResult < 0) {
            return binarySearch(target, data, first, middle);
        } else {
            return binarySearch(target, data, middle + 1, last);
        }
    }
```

Chapter	Section	No
9	6	1

Question

Complete the code for the `add` method.

Solution

```
//Solution to programming exercise 1, Section 6, Chapter 9
//File: \KW\CH09\SkipList.java startLine: 107 endLine 140
```

```
    /**
     * Inserts item where it belongs in the skip list.
     * @param item The item to be inserted
     * @return true If the item is inserted, false if the
     * item was already in the tree.
     */
    public boolean add(E item) {
        SLNode<E>[] update = search(item);
        if (update[0].links[0] != null
                && update[0].links[0].data.compareTo(item) == 0) {
            return false; // Item already in Skip List
        }
        // Increment size and adjust maxLevel
        size++;
        if (size > maxCap) {
            maxLevel++;
            maxCap = computeMaxCap(maxLevel);
            head.links = Arrays.copyOf(head.links, maxLevel);
            update = Arrays.copyOf(update, maxLevel);
            update[maxLevel - 1] = head;
        }
        // Create new node for item
        SLNode<E> newNode = new SLNode<E>(logRandom(), item);
        // Splice new node into list
        for (int i = 0; i < newNode.links.length; i++) {
            newNode.links[i] = update[i].links[i];
            update[i].links[i] = newNode;
        }
        return true;
    }
```

Chapter	Section	No
10	2	1

Question

Implement the `Edge` class.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter 10
//File: \KW\CH10\Edge.java startLine: 0 endLine 108
package KW.CH10;

/** An Edge represents a relationship between two
 *  vertices.
 *  @author Koffman and Wolfgang
 */
public class Edge {
    // Data Fields

    /** The source vertix */
    private int source;
    /** The destination vertix */
    private int dest;
    /** The weight */
    private double weight;

    // Constructor
    /** Construct an Edge with a source of from
     *  and a destination of to. Set the weight
     *  to 1.0.
     *  @param from - The source vertix
     *  @param to - The destination vertix
     */
    public Edge(int source, int dest) {
        this.source = source;
        this.dest = dest;
        weight = 1.0;
    }

    /** Construct a weighted edge with a source
     *  of from and a destination of to. Set the
     *  weight to w.
     *  @param from - The source vertix
     *  @param to - The destination vertix
     *  @param w - The weight
     */
    public Edge(int source, int dest, double w) {
        this.source = source;
        this.dest = dest;
        weight = w;
    }

    // Methods
    /** Get the source
     *  @return The value of source
     */
    public int getSource() {
        return source;
    }

    /** Get the destination
     *  @return The value of dest
     */
    public int getDest() {
        return dest;
    }

    /** Get the weight
     *  @return the value of weight
     */
    public double getWeight() {
        return weight;
    }

    /** Return a String representation of the edge
     *  @return A String representation of the edge
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("[(");
        sb.append(Integer.toString(source));
        sb.append(", ");
        sb.append(Integer.toString(dest));
        sb.append("): ");
        sb.append(Double.toString(weight));
        sb.append("]");
        return sb.toString();
    }

    /** Return true if two edges are equal. Edges
     *  are equal if the source and destination
```

```
         *  are equal. Weight is not conidered.
         *  @param obj The object to compare to
         *  @return true if the edges have the same source
         *  and destination
         */
        @Override
        public boolean equals(Object obj) {
            if (obj instanceof Edge) {
                Edge edge = (Edge) obj;
                return (source == edge.source && dest == edge.dest);
            } else {
                return false;
            }
        }

        /** Return a hash code for an edge.  The hash
         *  code is the source shifted left 16 bits
         *  exclusive or with the dest
         *  @return a hash code for an edge
         */
        @Override
        public int hashCode() {
            return (source << 16) ^ dest;
        }
}
```

Chapter	Section	No
10	3	1

Question

Implement the `loadEdgesFromFile` method for class `AbstractGraph`. If there are two values on a line, an edge with the default weight of 1.0 is inserted; if there are three values, the third value is the weight.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter 10
//File: \KW\CH10\AbstractGraph.java startLine: 62 endLine 78
    public void loadEdgesFromFile(Scanner scan)
        String line;
        while (scan.hasNextLine()) {
            line = scan.nextLine();
            String[] tokens = line.split("\\s+");
            int source = Integer.parseInt(tokens[0]);
            int dest = Integer.parseInt(tokens[1]);
            double weight = 1.0;
            if (tokens.length == 3) {
                weight = Double.parseDouble(tokens[2]);
            }
            insert(new Edge(source, dest, weight));
        }
    }
```

Chapter	Section	No
10	4	1

Question

Provide all accessor methods for class `DepthFirstSearch` and the constructor that specifies the order of start vertices.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter 10
//File: \KW\CH10\DepthFirstSearch.java startLine: 51 endLine 77
    /**
     * Construct the depth-first search of a Graph
     * selecting the start vertices in the specified order.
     * The first vertex visited is order[0].
     * @param graph The graph
     * @param order The array giving the order
     * in which the start vertices should be selected
     */
    public DepthFirstSearch(Graph graph, int[] order) {
        this.graph = graph;
        int n = graph.getNumV();
        parent = new int[n];
        visited = new boolean[n];
        discoveryOrder = new int[n];
        finishOrder = new int[n];
        for (int i = 0; i < n; i++) {
            parent[i] = -1;
        }
```

```
        for (int i = 0; i < n; i++) {
            if (!visited[order[i]]) {
                depthFirstSearch(i);
            }
        }
    }
//Solution to programming exercise 1, Section 4, Chapter 10
//File: \KW\CH10\DepthFirstSearch.java startLine: 105 endLine 127
    /** Get the finish order
     *  @return finish order
     */
    public int[] getFinishOrder() {
        return finishOrder;
    }

    /** Get the discovery  order
     *  @return discovery order
     */
    public int[] getDiscoveryOrder() {
        return discoveryOrder;
    }

    /** Get the parent
     *  @return the parent
     */
    public int[] getParent() {
        return parent;
    }
```

Chapter	Section	No
A	3	1

Question

Rewrite the **`for`** statement in Example A.4 using a **`do ... while`** loop.

Solution

```
//Solution to programming exercise 1, Section 3, Chapter A
//File: \KW\AXA\ExerciseA_3_1.java startLine: 16 endLine 27
        int nextInt = 1;
        do {
            if (nextInt % 2 == 0) {
                sum += nextInt;
            } else {
                prod *= nextInt;
            }
            nextInt++;
        } while (nextInt <= maxValue + 1);
```

Chapter	Section	No
A	4	1

Question

Write a Java program that displays all odd powers of 2 between 1 and 29. Display the power that 2 is being raised to, as well as the result, on each line. Use tab characters between numbers.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter A
//File: \KW\AXA\Exercise_A_4_1.java startLine: 13 endLine 21
    public static void main(String[] args) {
        System.out.println("n \tpower of 2");
        for (int i = 1; i < 30; i += 2) {
            System.out.println(i + "\t" + Math.pow(2, i));
        }
    }
```

Chapter	Section	No
A	5	1

Question

Write statements to extract the individual tokens in a string of the form `"Doe, John 5/15/65"`. Use the `indexOf` method to find the string `", "` and the symbol `/` and use the substring method to extract the substrings between these delimiters.

Solution

```
//Solution to programming exercise 1, Section 5, Chapter A
//File: \KW\AXA\ExerciseA_5_1.java startLine: 15 endLine 26
        int posComma = s.indexOf(", ");
```

```
        int posSpace = s.indexOf(' ', posComma + 2);
        int posSlash1 = s.indexOf('/', posSpace + 1);
        int posSlash2 = s.indexOf('/', posSlash1 + 1);
        String token1 = s.substring(0, posComma);
        String token2 = s.substring(posComma + 2, posSpace);
        String token3 = s.substring(posSpace + 1, posSlash1);
        String token4 = s.substring(posSlash1 + 1, posSlash2);
        String token5 = s.substring(posSlash2 + 1);
```

Chapter	Section	No
A	5	3

Question

For Self-Check Exercise 4, write a loop to display all the tokens that are extracted.

Solution

```
//Solution to programming exercise 3, Section 5, Chapter A
//File: \KW\AXA\ExerciseA_5_3.java startLine: 16 endLine 21
        for (String token : tokens) {
            System.out.println(token);
        }
```

Chapter	Section	No
A	6	1

Question

Write statements that double the value stored in the `Integer` object referenced by `i1`. Draw a diagram showing the objects referenced by `i1` before and after these statements execute.

Solution

```
//Solution to programming exercise 1, Section 6, Chapter A
//File: \KW\AXA\ExercisesA_6.java startLine: 20 endLine 23
        i1 = i1 * 2;
```

Chapter	Section	No
A	7	1

Question

Write a method `getInitials` that returns a string representing a Person object's initials. There should be a peroid after each initial. Write Javadoc tags for the method.

Solution

```
//Solution to programming exercise 1, Section 7, Chapter A
//File: \KW\AXA\Person2.java startLine: 201 endLine 211
    /**
     * Return the Person's initials as a string. There should be a peroid
     * after each initial.
     * @return the Person's initials as a string.
     */
    public String getInitials() {
        return givenName.charAt(0) + "." + familyName.charAt(0) + ".";
    }
```

Chapter	Section	No
A	7	3

Question

Write a method `compareTo` that compares two `Person` objects and returns an appropriate result based on comparison of the `ID` numbers. That is, if the `ID` number of the object that `compareTo` is applied to is less than (is greater than) the `ID` number of the argument object, the result should be negative (positive). The result should be 0 if they have the same `ID` numbers. Write Javadoc tags for the method.

Solution

```
//Solution to programming exercise 3, Section 7, Chapter A
//File: \KW\AXA\Person2.java startLine: 179 endLine 189
    /**
     * Compare this Person object to another Person object.  If this Person's
     * ID is less (greater) than the other Person's ID return a negative
     * (positive) value. Otherwise return 0.
     */
    public int compareTo(Person2 other) {
        return IDNumber.compareTo(other.IDNumber);
    }
```

Chapter	Section	No
A	7	4

Question

Write a method `switchNames` that exchanges a `Person` object's given and family names. Write Javadoc tags for the method.

Solution

```
//Solution to programming exercise 4, Section 7, Chapter A
//File: \KW\AXA\Person2.java startLine: 190 endLine 200
    /**
     * Switch the given and family names of the person
     */
    public void switchNames() {
        String temp = givenName;
        givenName = familyName;
        familyName = temp;
    }
```

Chapter	Section	No
A	8	1

Question

Write code for a method

`public static boolean sameElements(int[] a, int[] b)`

that checks whether two arrays have the same elements in some order, with the same multiplicities. For example, two arrays

`121 144 19 161 19 144 19 11`

and

`11 121 144 19 161 19 144 19`

would be considered to have the same elements because 19 appears three times in each array, 144 appears twice in each array, and all other elements appear once in each array.

Solution

```
//Solution to programming exercise 1, Section 8, Chapter A
//File: \KW\AXA\ExerciseA_8_1.java startLine: 14 endLine 50
    /**
     * Method to determine if two arrays contain the same elements with the
     * same multiplicites.
     * @param a The first array
     * @param b The second array
     * @return true if b contains the same elements as a
     */
    public static boolean sameElements(int[] a, int[] b) {
        if (a.length != b.length) {
            return false;
        }
        for (int i = 0; i < a.length; i++) {
            if (countOccurances(a, a[i]) != countOccurances(b, a[i])) {
                return false;
            }
        }
        return true;
    }

    /**
     * Method to count the number of times a integer occurs in an array
     * @param x The array
     * @param t The targer
     * @return The number of times t occurs in x
     */
    public static int countOccurances(int x[], int t) {
        int count = 0;
        for (int i = 0; i < x.length; i++) {
            if (x[i] == t) {
                count++;
            }
        }
        return count;
    }
```

Chapter	Section	No
A	8	3

Question

For the two-dimensional array `letters` in Example A.19, assume `letters[i]` is going to be used to store an array that contains the

individual characters in `String` object `next`. Allocate storage for `letters[i]` based on the length of `next` and write a loop that stores each character of `next` in the corresponding element of `letters[i]`. For example, the first character in `next` should be stored in `letters[i][0]`.

Solution

```
//Solution to programming exercise 3, Section 8, Chapter A
//File: \KW\AXA\ExerciseA_8_3.java startLine: 15 endLine 21
        letters[i] = new char[next.length()];
        for (int j = 0; j < next.length(); j++) {
            letters[i][j] = next.charAt(j);
        }
```

Chapter	Section	No
A	9	1

Question

Write a main method that reads the data for two `Person` objects, creates the objects, and displays the objects and a message indicating whether they represent the same Person.

Solution

```
//Solution to programming exercise 1, Section 9, Chapter A
//File: \KW\AXA\ExerciseA_9_1.java startLine: 15 endLine 40
    public static Person readPerson() {
        String familyName;
        String givenName;
        String IDNumber;
        int birthYear;

        familyName = JOptionPane.showInputDialog("Enter family name");
        givenName = JOptionPane.showInputDialog("Enter given name");
        IDNumber = JOptionPane.showInputDialog("Enter the ID number");
        String birthYearString = JOptionPane.showInputDialog("Enter birth year");
        birthYear = Integer.parseInt(birthYearString);
        return new Person(familyName, givenName, IDNumber, birthYear);
    }

    public static void main(String[] args) {
        Person person1 = readPerson();
        Person person2 = readPerson();
        if (person1.equals(person2)) {
            System.out.println(person1 + " is the same as " + person2);
        } else {
            System.out.println(person1 + " is not the same as " + person2);
        }
    }
```

Chapter	Section	No
A	10	1

Question

Write a method for class `Person` that reads the data for a single employee from a `BufferedReader` object (the method argument). Assume there is one data item per line.

Solution

```
//Solution to programming exercise 1, Section 10, Chapter A
//File: \KW\AXA\Person.java startLine: 1 endLine 5
import java.io.BufferedReader;
import java.io.IOException;
//Solution to programming exercise 1, Section 10, Chapter A
//File: \KW\AXA\Person.java startLine: 150 endLine 174
    /**
     * Method to read a Person object from a buffered reader.  Each person
     * object is represented by four data lines containing the givenNeme,
     * familyName, IDNumber, and birthYear.
     * @param br The BufferedReader
     * @return a new Person object
     */
    public static Person readPerson(BufferedReader br) {
        try {
            String familyName = br.readLine();
            String givenName = br.readLine();
            String IDNumber = br.readLine();
            String birthYear = br.readLine();
            return new Person(familyName, givenName, IDNumber,
                Integer.parseInt(birthYear));
        } catch (IOException ioex) {
            ioex.printStackTrace();
            System.exit(1);
```

```
        }
        // should not get here
        return null;
    }
```

Chapter	Section	No
A	11	1

Question

For the try block

```
try {
    numStr = in.readLine();
    num = Integer.parseInt(numStr);
    average = total / num;
}
```

write a `try-catch-finally` sequence with `catch` clauses for `ArithmeticException`, `NumberFormatException`, and `IOException`. For class `ArithmeticException`, set `average` to zero and display an error message indicating the kind of exception, display the stack trace, and exit with an error indication. After exiting the **`try`** block or the **`catch`** block for `ArithmeticException`, display the message "That's all folks" in the **`finally`** block.

Solution

```
//Solution to programming exercise 1, Section 11, Chapter A
//File: \KW\AXA\ExerciseA_11_1.java startLine: 26 endLine 44
        } catch (IOException ioex) {
            System.err.println("IO Exception");
            ioex.printStackTrace();
            System.exit(1);
        } catch (NumberFormatException nfex) {
            System.err.println("Number Format Exception");
            nfex.printStackTrace();
            System.exit(1);
        } catch (ArithmeticException arex) {
            average = 0;
            System.err.println("Arithmetic exception");
            arex.printStackTrace();
            System.exit(1);
        } finally {
            System.out.println("That's all folks");
        }
```

Chapter	Section	No
A	12	1

Question

The syntax display for the **`throw`** statement had the following example:

```
throw new FileNotFoundException("File " + fileSource
+ " not found");
```

Write a **`catch`** clause for a method farther back in the call chain that handles this exception.

Solution

```
//Solution to programming exercise 1, Section 12, Chapter A
//File: \KW\AXA\ExerciseA_12_1.java startLine: 47 endLine 53
        } catch (FileNotFoundException ex) {
            System.err.println(ex.getMessage());
            System.exit(1);
        }
```

Chapter	Section	No
C	1	1

Question

Write statements to create a button with label Submit, and register class `DoSubmit` as an action listener for this button.

Solution

```
//Solution to programming exercise 1, Section 1, Chapter C
//File: \KW\AXC\ExerciseC_1.java startLine: 25 endLine 29
        JButton submit = new JButton("Submit");
        submit.addActionListener(new DoSubmit());
```

Chapter	Section	No
C	1	3

Question

Write class `DoSave` for the phone directory problem.

Solution

```
//Solution to programming exercise 3, Section 1, Chapter C
//File: \KW\AXC\PDButtonUI.java startLine: 154 endLine 169
    /** Class to respond to the Save button */
    private class DoSave implements ActionListener {

        /** Method to save the directory to the data file.
         *  Pre:  The directory has been loaded with data.
         *  Post: The current contents of the directory have been saved
         *  to the data file.
         */
        @Override
        public void actionPerformed(ActionEvent e) {
            theDirectory.save();
        }
    }
```

Chapter	Section	No
C	2	1

Question

Create a modified version of `TwoCircles` to show just the outline of a circle in green.

Solution

```
//Solution to programming exercise 1, Section 2, Chapter C
//File: \KW\AXC\MyCircle2.java startLine: 41 endLine 56
    /** Paint the component when it changes. This method is called
     *  by the Swing API.
     *  @param g The graphics object used for painting. */
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.YELLOW);
        g.fillRect(0, 0, size, size);
        if (showCircle) {
            g.setColor(Color.GREEN);
            g.drawOval(0, 0, size, size);
        }
    }
```

Chapter	Section	No
C	3	1

Question

Create a frame that has a grid of labels that looks like this:

yellow		white		blue
		green		
	orange		purple	
green		red		blue
yellow			orange	

Solution

```
//Solution to programming exercise 1, Section 3, Chapter C
//File: \KW\AXC\ExerciseC_3_1.java startLine: 11 endLine 44
public class ExerciseC_3_1 extends JFrame {

    private ExerciseC_3_1() {
        Border blackBorder = BorderFactory.createLineBorder(Color.BLACK);
        Color[][] colors = new Color[][]{
            {Color.YELLOW, Color.GRAY, Color.WHITE, Color.GRAY, Color.BLUE},
            {Color.GRAY, Color.GRAY, Color.GREEN, Color.GRAY, Color.GRAY},
            {Color.GRAY, Color.ORANGE, Color.GRAY, new Color(128, 0, 255), Color.GRAY},
            {Color.GREEN, Color.GRAY, Color.RED, Color.GRAY, Color.BLUE},
            {Color.YELLOW, Color.GRAY, Color.GRAY, Color.ORANGE, Color.GRAY}};
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(5, 5));
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                JLabel label = new JLabel();
                label.setBackground(colors[i][j]);
                label.setBorder(blackBorder);
                label.setOpaque(true);
```

```
                label.setPreferredSize(new Dimension(50, 25));
                panel.add(label);
            }
        }
        add(panel);
        pack();
    }

    public static void main(String[] args) {
        new ExerciseC_3_1().setVisible(true);
    }
}
```

Chapter	Section	No
C	3	3

Question

Create a frame that looks like Figure C.12 using nested `BoxLayout` managers.

Solution

```
//Solution to programming exercise 3, Section 3, Chapter C
//File: \KW\AXC\ExerciseC_3_3.java startLine: 10 endLine 38
public class ExerciseC_3_3 extends JFrame {

    public ExerciseC_3_3() {

        JPanel rows = new JPanel();
        Border blackBorder = BorderFactory.createLineBorder(Color.BLACK);
        rows.setLayout(new BoxLayout(rows, BoxLayout.Y_AXIS));
        for (int i = 0; i < 5; i++) {
            JPanel row = new JPanel();
            row.setLayout(new BoxLayout(row, BoxLayout.X_AXIS));
            for (int j = 0; j < 10; j++) {
                JLabel label = new JLabel(i + ", " + j);
                label.setBorder(blackBorder);
                row.add(label);
            }
            rows.add(row);
        }
        add(rows);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        pack();
    }

    public static void main(String[] args) {
        new ExerciseC_3_3().setVisible(true);
    }
}
```

Chapter	Section	No
C	4	1

Question

A GUI that displays three button groups, containing radio buttons and a text area, is shown in Figure C.19. Whenever one of the radio buttons is changed, the text should be updated. Write the program.

Solution

```
//Solution to programming exercise 1, Section 4, Chapter C
//File: \KW\AXC\RadioButtonExercise.java startLine: 0 endLine 70
package KW.AXC;

import java.awt.Container;
import java.awt.FlowLayout;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class RadioButtonExercise extends JFrame {

    private RadioButtonGroupPanel temperaturePanel;
    private RadioButtonGroupPanel conditionPanel;
    private RadioButtonGroupPanel visibilityPanel;
    private JTextArea textArea = new JTextArea(4, 34);

    public static void main(String[] args) {
```

```
        new RadioButtonExercise();
    }

    private RadioButtonExercise() {
        PropertyChangeListener updateDisplays = new PropertyChangeListener() {

            public void propertyChange(PropertyChangeEvent e) {
                StringBuffer sb =
                        new StringBuffer("Here are the weather conditions\n");
                sb.append(temperaturePanel.getCurrentChoice());
                sb.append("\n");
                sb.append(conditionPanel.getCurrentChoice());
                sb.append("\n");
                sb.append(visibilityPanel.getCurrentChoice());
                textArea.setText(sb.toString());
            }
        };
        String[] temperatures = {"Cold", "Mild", "Hot"};
        temperaturePanel =
                new RadioButtonGroupPanel(temperatures);
        temperaturePanel.addPropertyChangeListener("CurrentChoice",
                updateDisplays);
        String[] conditions = {"Dry", "Wet"};
        conditionPanel =
                new RadioButtonGroupPanel(conditions);
        conditionPanel.addPropertyChangeListener("CurrentChoice",
                updateDisplays);
        String[] visibilities = {"Clear", "Rain", "Snow", "Mix"};
        visibilityPanel =
                new RadioButtonGroupPanel(visibilities);
        visibilityPanel.addPropertyChangeListener("CurrentChoice",
                updateDisplays);
        JPanel buttonPanels = new JPanel();
        buttonPanels.setLayout(new BoxLayout(buttonPanels,
                BoxLayout.Y_AXIS));
        buttonPanels.add(temperaturePanel);
        buttonPanels.add(conditionPanel);
        buttonPanels.add(visibilityPanel);
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        contentPane.add(buttonPanels);
        updateDisplays.propertyChange(null);
        contentPane.add(textArea);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setTitle("Radio Button Exercise");
        setSize(400, 200);
        setVisible(true);
    }
}
//Solution to programming exercise 1, Section 4, Chapter C
//File: \KW\AXC\RadioButtonGroupPanel.java startLine: 0 endLine 56
package KW.AXC;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BoxLayout;
import javax.swing.ButtonGroup;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

public class RadioButtonGroupPanel extends JPanel {

    // Data Fields
    String[] selections = null;
    JRadioButton[] radioButtons = null;
    ButtonGroup buttonGroup = null;
    String currentChoice = null;

    // Constructor
    public RadioButtonGroupPanel(String[] selections) {
        this.selections = selections;
        radioButtons = new JRadioButton[selections.length];
        setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
        // Create a button group for the buttons
        buttonGroup = new ButtonGroup();
        // Create radio buttons and add them to the panel
        // Also add them to the button group
        ActionListener newSelection = new SelectionChangeMade();
        for (int i = 0; i != selections.length; ++i) {
            radioButtons[i] = new JRadioButton(selections[i]);
            radioButtons[i].getModel().setActionCommand(selections[i]);
            radioButtons[i].addActionListener(newSelection);
            buttonGroup.add(radioButtons[i]);
```

```
            add(radioButtons[i]);
        }
        radioButtons[0].setSelected(true);
        currentChoice = selections[0];
    }

    // Action Listener Classes
    private class SelectionChangeMade implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            String oldChoice = currentChoice;
            currentChoice =
                    buttonGroup.getSelection().getActionCommand();
            firePropertyChange("CurrentChoice", oldChoice, currentChoice);
        }
    }

    public String getCurrentChoice() {
        return currentChoice;
    }
}
```

Chapter	Section	No
C	5	1

Question

Write a distance converter that converts from miles to kilometers. There are 1.609344 kilometers in a mile. Display the result with only one fraction digit.

Solution

```
//Solution to programming exercise 1, Section 5, Chapter C
//File: \KW\AXC\DistanceConverter.java startLine: 0 endLine 30
package KW.AXC;

/** DistanceConverter is a class with static methods
 *  that convert between miles and kilometers.
 *  @author Koffman & Wolfgang
 */
public class DistanceConverter {

    /** The number of kilometers in a mile.
     */
    private static final double KILOMETERS_PER_MILE = 1.609344;

    /** Convert a value in kilometers to miles.
     *  @param kilometers The value in kilometers
     *  @return The value in miles
     */
    public static double toMiles(double kilometers) {
        return kilometers / KILOMETERS_PER_MILE;
    }

    /** Convert a value in miles to kilometers.
     *  @param miles The value in miles
     *  @return The value in kilometers
     */
    public static double toKilometers(double miles) {
        return miles * KILOMETERS_PER_MILE;
    }
}
//Solution to programming exercise 1, Section 5, Chapter C
//File: \KW\AXC\DistanceConverterGUI.java startLine: 0 endLine 103
package KW.AXC;

import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

/** VolumeConverterGUI is a GUI application that converts
 *  values in miles to kilometers and vice versa.
 *  @author Koffman & Wolfgang
 */
public class DistanceConverterGUI
        extends JFrame {
```

```
    // Data Fields
    /** Text field to hold miles value */
    private JTextField milesField;
    /** Text field to hold kilometers value */
    private JTextField kilometersField;

    // Main Method
    /** Create an instance of the application and show it.
     *  @param args Command Line Arguments -- not used
     */
    public static void main(String[] args) {
        JFrame frame = new DistanceConverterGUI();
        frame.setVisible(true);
    }

    // Constructor
    /** Construct the components and add them to the frame. */
    public DistanceConverterGUI() {
        super("Distance Converter");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        // Get a reference to the content pane.
        Container contentPane = getContentPane();
        // Set the layout manager to grid layout.
        contentPane.setLayout(new GridLayout(2, 2));
        contentPane.add(new JLabel("miles"));
        milesField = new JTextField(15);
        milesField.addActionListener(new NewMilesValue());
        contentPane.add(milesField);
        contentPane.add(new JLabel("kilometers"));
        kilometersField = new JTextField(15);
        kilometersField.addActionListener(new NewKilometersValue());
        contentPane.add(kilometersField);
        // Size the frame to fit.
        pack();
    }

    // Inner Classes
    /** Class to respond to new miles value. */
    private class NewMilesValue implements ActionListener {

        /** Convert the miles value to corresponding kilometers value.
         *  @param e ActionEvent object - not used
         */
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                double milesValue =
                        Double.parseDouble(milesField.getText());
                double kilometersValue =
                        DistanceConverter.toKilometers(milesValue);
                kilometersField.setText(String.format("%.1f", kilometersValue));
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null,
                        "Invalid Number Format",
                        "",
                        JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    /** Class to respond to new kilometers value. */
    private class NewKilometersValue implements ActionListener {

        /** Convert the kilometers value to corresponding miles value.
         *  @param e ActionEvent object - not used
         */
        @Override
        public void actionPerformed(ActionEvent e) {
            try {
                double kilometersValue =
                        Double.parseDouble(kilometersField.getText());
                double milesValue =
                        DistanceConverter.toMiles(kilometersValue);
                milesField.setText(String.format("%.1f", milesValue));
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null,
                        "Invalid Number Format",
                        "",
                        JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

Chapter	Section	No
C	6	1

Question

Code the methods `getWidth`, `getHeight`, `getName`, `getMnemonic`, `getShortCut`, and `getIcon` for the `DrawableShape` classes defined in Section 3.7.

Solution

```
//Solution to programming exercise 1, Section 6, Chapter C
//File: \KW\AXC\drawapp\DrawableCircle.java startLine: 72 endLine 96
    @Override
    public int getWidth() {
        return (int)(2 * getRadius());
    }

    @Override
    public int getHeight() {
        return (int)(2 * getRadius());
    }

    @Override
    public String getName() {
        return NAME;
    }

    @Override
    public Icon getIcon(int size) {
        ((Circle) theShape).setRadius(size / 2);
        pos.x = size / 2;
        pos.y = size / 2;
        return new DrawableFigureIcon(this);
    }
//Solution to programming exercise 1, Section 6, Chapter C
//File: \KW\AXC\drawapp\DrawableRectangle.java startLine: 82 endLine 122
    /** Get the width of the rectangle
     *  @return The width of the rectangle
     */
    @Override
    public int getWidth() {
        return ((Rectangle) theShape).getWidth();
    }

    /** Get the height of the rectangle
     *  @return the height of the rectangle
     */
    @Override
    public int getHeight() {
        return ((Rectangle) theShape).getHeight();
    }

    /** Create an icon of a DrawableRectanle. The icon is a Drawable
     *  rectangle of width equal to the size and height equal to 3/4
     *  the size, and whose origin is at 0, 0
     *  @param size The size of the bounding rectangle.
     *  @return an icon of a DrawableRectangle
     */
    @Override
    public Icon getIcon(int size) {
        Rectangle rectangle = (Rectangle) theShape;
        rectangle.setWidth(size);
        rectangle.setHeight(3 * size / 4);
    setPoint(new Point(0, 0));
    return new DrawableFigureIcon(this);
  }


  /** Get the name of this shape
   *  @return the name of this shape
   */
    @Override
    public String getName() {
        return NAME;
    }
//Solution to programming exercise 1, Section 6, Chapter C
//File: \KW\AXC\drawapp\DrawableTriangle.java startLine: 77 endLine 104
    /** Accessors */
    @Override
    public int getWidth() {
```

```
        return ((RtTriangle) theShape).getBase();
    }

    @Override
    public int getHeight() {
        return ((RtTriangle) theShape).getHeight();
    }

    @Override
    public String getName() {
        return NAME;
    }

    @Override
    public Icon getIcon(int size) {
        RtTriangle triangle = (RtTriangle) theShape;
        triangle.setBase(size);
       triangle.setHeight(size);
       pos.x = 0;
       pos.y = size;
       return new DrawableFigureIcon(this);
   }
```

Chapter	Section	No
C	6	3

Question

Code method `setBorderColor`.

Solution

```
//Solution to programming exercise 3, Section 6, Chapter C
//File: \KW\AXC\drawapp\DrawApp.java startLine: 220 endLine 243
    /** ActionListener class for the SetBorderColor command */
    private class SetBorderColor implements ActionListener {

        /** Action Performed is to display the color chooser
         *  dialog and then set the border color of each of the
         *  figure prototypes. The display is then repainted
         *  @param e ActionEvent -- Not Used
         */
        @Override
        public void actionPerformed(ActionEvent e) {
            currentBorderColor =
                    JColorChooser.showDialog(
                    DrawApp.this,
                    "Select Border Color",
                    currentBorderColor);
            for (int i = 0; i < theFigKinds.length; i++) {
                theFigKinds[i].setBorderColor(currentBorderColor);
            }
            repaint();
        }
    }
```

Chapter	Section	No
C	7	1

Question

Code the `stretchMe` method for the `DrawableShape` classes in Section 3.6.

Solution

```
//Solution to programming exercise 1, Section 7, Chapter C
//File: \KW\AXC\drawapp\DrawableCircle.java startLine: 41 endLine 61
    @Override
    public void stretchMe(Graphics g, Point stretchPoint,
            Color background) {
        g.setColor(borderColor);
        g.setXORMode(background);
        if (stretchableOutlineDrawn) {
            BoundingRectangle br = getBoundingRectangle();
            g.drawOval(br.x, br.y, br.w, br.h);
        }
        double deltaX = pos.x - stretchPoint.x;
        double deltaY = pos.y - stretchPoint.y;
        ((Circle) theShape).setRadius((int) Math.sqrt(deltaX * deltaX
                + deltaY * deltaY));
        BoundingRectangle br = getBoundingRectangle();
        g.drawOval(br.x, br.y, br.w, br.h);
        stretchableOutlineDrawn = true;
```

```
        g.setPaintMode();
    }
//Solution to programming exercise 1, Section 7, Chapter C
//File: \KW\AXC\drawapp\DrawableRectangle.java startLine: 48 endLine 72
    /** Erases the previously drawn outline, re-sizes the figure
     *  based upon the stretch point, and draws the new outline
     *  @param g The graphics context
     *  @param stretchPoint The current mouse poisition
     *  @param background The current background color
     */
    public void stretchMe(Graphics g, Point stretchPoint, Color background) {
        int x, y, w, h;
        g.setColor(borderColor);
        g.setXORMode(background);
        if (stretchableOutlineDrawn) {
            BoundingRectangle br = getBoundingRectangle();
            g.drawRect(br.x, br.y, br.w, br.h);
        }
        Rectangle rectangle = (Rectangle) theShape;
        rectangle.setWidth(stretchPoint.x - pos.x);
        rectangle.setHeight(stretchPoint.y - pos.y);
        BoundingRectangle br = getBoundingRectangle();
        g.drawRect(br.x, br.y, br.w, br.h);
        stretchableOutlineDrawn = true;
        g.setPaintMode();
    }
//Solution to programming exercise 1, Section 7, Chapter C
//File: \KW\AXC\drawapp\DrawableTriangle.java startLine: 54 endLine 71
    @Override
    public void stretchMe(Graphics g, Point stretchPoint, Color background) {
        g.setColor(borderColor);
        g.setXORMode(background);
        Polygon tri = makePolygon();
        if (stretchableOutlineDrawn) {
            g.drawPolygon(tri);
        }
        RtTriangle triangle = (RtTriangle) theShape;
        triangle.setBase(stretchPoint.x - pos.x);
        triangle.setHeight(pos.y - stretchPoint.y);
        tri = makePolygon();
        g.drawPolygon(tri);
        stretchableOutlineDrawn = true;
    }
```

Chapter	Section	No
C	7	3

Question

Create an anonymous listener class for class ComboBoxDemo shown in Listing C.11.

Solution

```
//Solution to programming exercise 3, Section 7, Chapter C
//File: \KW\AXC\ComboBoxDemo2.java startLine: 33 endLine 42
        comboBox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String choice = (String) comboBox.getSelectedItem();
                System.out.println(choice + " is selected");
            }
        });
```