Data Structure Assignment #2

20171620 문성찬

1. Source Program

1) BagInterface

```java
/**
    An interface that describes the operations of a bag of objects.
    @author 20171620 MoonSeongchan
 */

public interface BagInterface<T> {

    /** Gets the current number of entries in this bag.
     @return The integer number of entries currently in the bag */
    public int getCurrentSize();

    /** Sees whether this bag is empty.
        @return True if the bag is empty, or false if not. */
    public boolean isEmpty();

    /** Adds a new entry to this bag.
        @param newEntry The object to be added as a new entry.
        @return True if the addition is successful, or false if not. */
    public boolean add(T newEntry);

    /** Removes one unspecified entry from this bag, if possible.
        @return Either the removed entry, if the removal was successful, or null
*/
    public T remove();

    /** Removes one occurrence of a given entry from this bag, if possible.
        @param anEntry The entry to be removed.
        @return True if the removal was successful, or false if not. */
    public boolean remove (T anEntry);

    /** Removes all entries from this bag */
    public void clear();


    /** Counts the number of times a given entry appears in this bag.
        @param anEntry The entry to be counted.
        @return The number of times anEtry appears in the bag. */
    public int getFrequencyOf(T anEntry);


    /** Tests whether this bag contains a given entry.
        @param anEntry The entry to locate.
        @return True if the bag contains anEntry, of false if not. */
    public boolean contains(T anEntry);


    /** Retrieves all entries that are in this bag.
        @return A newly allocated array of all the entries in the bag.
        Note: If the bag is empty, the returned array is empty. */
    public T[] toArray();

}   // end BagInterface
```

2) ArrayBag

BagInterface를 implements하는 ArrayBag 구현

```java
import java.util.Arrays;

/**
 * A class of bags whose entries are stored in a fixed-size array.
 * @author 20171620 MoonSeongchan
 */
public final class ArrayBag<T> implements BagInterface<T> {

    private T[] bag;
    private int numberOfEntries;
    private static final int DEFAULT_CAPACITY = 25;
    private boolean initialized = false;
    private static final int MAX_CAPACITY = 10000;

    /** Creates an empty bag whose capacity is 25. */
    public ArrayBag() {
        this(DEFAULT_CAPACITY);
    }

    /** Creates an empty bag having a given capacity
        @param desiredCapacity The integer capacity desired.
     */
    public ArrayBag(int desiredCapacity) {
        if (desiredCapacity <= MAX_CAPACITY) {
            // The cast is safe because the new array contains null entries.
            @SuppressWarnings("unchecked")
            T[] tempBag = (T[])new Object[desiredCapacity];
            bag = tempBag;
            numberOfEntries = 0;
            initialized = true;
        }else {
            throw new IllegalStateException("Attempt to create a bag whose
capacity exceeds allowed maximum");

        }
    }

    /** Throws an exception if this object is not initialized. */
    private void checkInitialization() {
        if (!initialized) {
            throw new IllegalStateException("Attempt to create a bag whose
capacity exceeds allowed maximum");
```

```
        }
    }

    /** Doubles the size of the array bag.
     * Precondition: checkInitialization has been called.
     */
    private void doubleCapacity() {
        int newLength = 2 * bag.length;
        checkCapacity(newLength);
        bag = Arrays.copyOf(bag,newLength);
    }

    // Throws an exception if the client requests a capacity that is too large.
    private void checkCapacity(int capacity) {
        if (capacity > MAX_CAPACITY) {
            throw new IllegalStateException("Attempt to create a bag whose
capacity exceeds allowed maximum of "+MAX_CAPACITY);
        }
    }

    /** Adds a new entry to this bag.
        @param newEntry The object to be added as a new entry.
        @return True. */
    public boolean add(T newEntry)
    {
        checkInitialization();
        if (isArrayFull()) {
            doubleCapacity();
        }
        bag[numberOfEntries] = newEntry;
        numberOfEntries++;

        return true;
    }

    /** Returns true if the arraybag is full, or false if not. */
    private boolean isArrayFull() {
        return numberOfEntries >= bag.length;
    }

    /** Retrieves all entries that are in this bag.
     * @return A newly allocated array of all the entries in the bag. */
    public T[] toArray() {
        // The cast is safe because the new array contains null entries.
        @SuppressWarnings("unchecked")
        T[] result = (T[])new Object[numberOfEntries];
        for (int index = 0; index < numberOfEntries; index ++) {
            result[index] = bag[index];
        }
        return result;
    }

    /** Returns True if the bag is empty, or false if not. */
    public boolean isEmpty() {
        return numberOfEntries == 0;
    }

    /** Returns current size of bag */
    public int getCurrentSize() {
        return numberOfEntries;
    }

    /** Counts the number of times a given entry appears in this bag.
     * @param anEntry The entry to be counted.
     * @return The number of times anEntry appears in the bag.
     */
    public int getFrequencyOf(T anEntry) {
        checkInitialization();
        int counter = 0;

        for (int index =0;index < numberOfEntries; index ++) {
            if (anEntry.equals(bag[index])) {
                counter ++;
            }
        }
        return counter;
    }

    /** Check if there is anEntry in this bag.
     * @param anEntry The entry to be searched.
     * @return boolean value the result of search.
     */
    public boolean contains(T anEntry) {
        checkInitialization();
        return getIndexOf(anEntry) > -1;
    }

    /** Removes all entries from this bag. */
    public void clear() {
        while(!isEmpty()) {
            remove();
        }
    }

    /** Removes and returns the entry at a given index within the array bag.
     * @param givenIndex
     * @return entry to be removed, if no such entry exists, returns null.
     */
    private T removeEntry(int givenIndex) {
        T result = null;
        if (!isEmpty() && (givenIndex >= 0)) {
            result = bag[givenIndex];
            bag[givenIndex] = bag[numberOfEntries-1];
            bag[numberOfEntries-1] = null;
            numberOfEntries--;
        }
        return result;
    }

    /** Locates a given entry within the array bag.
     * @param anEntry The entry to be removed.
     * @return the index of the entry, if located, or -1 otherwise. */
    private int getIndexOf(T anEntry) {
        int where = -1;
        boolean found = false;
        int index = 0;
        while(!found && (index < numberOfEntries)) {
            if (anEntry.equals(bag[index])) {
                found = true;
                where = index;
            }
            index++;
        }
        return where;
    }

    /** Remove one unspecified entry from this bag, if possible
     * @return Either the removed entry, if the removal was successful, or null
otherwise.
     */
    public T remove() {
        checkInitialization();
        T result =  removeEntry(numberOfEntries-1);
        return result;
    }

    /** Removes one occurrence of a given entry from this bag, if possible.
     * @param anEntry The entry to be removed.
     * @return True if the removal was successful, or false if not.
     */
    public boolean remove(T anEntry) {
        checkInitialization();
        int index = getIndexOf(anEntry);
        T result = removeEntry(index);
        return anEntry.equals(result);
    }
}
```

3) LinkedBag

BagInterface를 implements하는 LinkedBag 구현

```
/**
    A class of bags whose entries are stored in a chain of linked nodes.
    The bag is never full.
    @author 20171620 MoonSeongchan
 */
public final class LinkedBag<T> implements BagInterface<T>{

    private Node firstNode;     // Reference to first node
    private int numberOfEntries;

    /** Creates an empty bag whose firstNode is null. */
    public LinkedBag() {
        firstNode = null;
        numberOfEntries = 0;
    }   // end default constructor

    /** Adds a new entry to this bag.
     * @param newEntry The object to be added as a new entry.
     * @return True.     */
    public boolean add(T newEntry) {
        // Add to beginning of chain:
```

```java
        Node newNode = new Node(newEntry);
        newNode.setNextNode(firstNode);
        firstNode = newNode;    // New node is at beginning of chain
        numberOfEntries++;
        return true;
    }   // end add

    /** Retrieves all entries that are in this bag.
     * @return A newly allocated array of all the entries in the bag.
     */
    public T[] toArray() {
        //The cast is safe because the new array contains null entries
        @SuppressWarnings("unchecked")
        T[] result = (T[])new Object[numberOfEntries];  // Unchecked cast

        int index = 0;
        Node currentNode = firstNode;
        while ((index<numberOfEntries) && (currentNode != null)) {
            result[index] = currentNode.data;
            index++;
            currentNode = currentNode.getNextNode();
        }   // end while

        return result;
    }   // end toArray

    /** Counts the number of times a given entry appears in this bag.
     * @param anEntry The entry to be counted.
     * @return The number of times anEntry appears in the bag.
     */
    public int getFrequencyOf(T anEntry) {
        int frequency = 0;
        int loopCounter = 0;
        Node currentNode = firstNode;

        while((loopCounter<numberOfEntries)&&(currentNode != null)) {
            if (anEntry.equals(currentNode.data)) {
                frequency++;
            loopCounter++;
            currentNode = currentNode.getNextNode();

            }
        }
        return frequency;
    }

    /** Check if there is anEntry in this bag.
     * @param anEntry The entry to be searched.
     * @return boolean value the result of search.
     */
    public boolean contains(T anEntry) {
        boolean found = false;
        Node currentNode = firstNode;

        while(!found && (currentNode != null)) {
            if (anEntry.equals(currentNode.data)) {
                found = true;
            }else {
                currentNode = currentNode.getNextNode();
            }
        }
        return found;
    }

    /** Remove one unspecified entry from this bag, if possible
     * @return Either the removed object, if the removal was successful, or
null.
     */
    public T remove() {
        T result = null;
        if (firstNode != null) {
            result = firstNode.getData();
            firstNode = firstNode.next;
            numberOfEntries--;
        }
        return result;
    }

    /** Locates a given entry within this bag.
     *  Returns a reference to the node containing the entry, if located, or
null otherwise.
     */
    private Node getReferenceTo(T anEntry) {
        boolean found = false;
        Node currentNode = firstNode;

        while(!found && (currentNode != null)) {
            if (anEntry.equals(currentNode.data)) {
                found = true;
            }else {
                currentNode = currentNode.getNextNode();
            }
        }
        return currentNode;
    }

    /** Removes one occurrence of a given entry from this bag, if possible.
     *  @param anEntry The entry to be removed.
     *  @return True if the removal was successful, or false otherwise.
     */
    public boolean remove(T anEntry) {
        boolean result = false;
        Node nodeN = getReferenceTo(anEntry);

        if(nodeN != null) {
            nodeN.data = firstNode.data;
            firstNode = firstNode.next;
            numberOfEntries--;
            result = true;
        }
        return result;
    }

    /** Returns True if the bag is empty, or false if not. */
    public boolean isEmpty() {
        return numberOfEntries == 0;
    }

    /** Returns current size of bag */
    public int getCurrentSize() {
        return numberOfEntries;
    }

    /** Removes all entries from this bag. */
    public void clear() {
        while(!isEmpty()) {
            remove();
        }
    }

    private class Node{
        private T data;     // Entry in bag
        private Node next;  // Link to next Node

        private Node(T dataPortion) {
            this(dataPortion,null);
        }   // end default constructor

        private Node(T dataPortion, Node nextNode) {
            data = dataPortion;
            next = nextNode;
        }   // end constructor

        private T getData() {
            return data;
        }

        private void setData(T newData) {
            data = newData;
        }

        private Node getNextNode() {
            return next;
        }

        private void setNextNode(Node nextNode) {
            next = nextNode;
        }
    }   // end Node
}
```

4) PayrollSystemTest_ArrayBag

ArrayBag을 사용해 PayrollSystemTest 구현

```
// Employee hierarchy test program.
import java.util.Scanner; // program uses Scanner to obtain user input

public class PayrollSystemTest_ArrayBag
{
    public static void main( String[] args )
    {
        BagInterface<Employee> aBag = new ArrayBag<>();

        // create subclass objects
        SalariedEmployee salariedEmployee =
           new SalariedEmployee(
           "John", "Smith", "111-11-1111", 1, 15, 1944, 2,1,2000, 400.00 );
        SalariedEmployee salariedEmployee1 =
                new SalariedEmployee(
                "Sam", "Kim", "111-12-3411", 2, 15, 1947, 4,17,2001, 600.00 );
        SalariedEmployee salariedEmployee2 =
                new SalariedEmployee(
                "Dan", "David", "351-13-1251", 3, 25, 1954, 4,7,2002, 800.00 );
        HourlyEmployee hourlyEmployee =
                new HourlyEmployee(
                "Karen", "Price", "222-22-2222", 4, 29, 1960,3,6,2003 ,12.75, 40
);
        HourlyEmployee hourlyEmployee1 =
                new HourlyEmployee(
                "Lim", "Deep", "462-22-2522", 5, 29, 1940,2,6,2004 ,14.75, 40 );
        HourlyEmployee hourlyEmployee2 =
                new HourlyEmployee(
                "Koo", "Cold", "364-23-2352", 6, 29, 1960,3,6,2005 ,16.75, 40 );
        CommissionEmployee commissionEmployee =
                new CommissionEmployee(
                "Sue", "Jones", "333-33-3333", 7, 8, 1954,7,3,2006, 5000, .06 );
        CommissionEmployee commissionEmployee1 =
                new CommissionEmployee(
                "Soo", "Kim", "333-33-1113", 8, 8, 1954,7,3,2007, 10000, .06 );

        BasePlusCommissionEmployee basePlusCommissionEmployee =
                new BasePlusCommissionEmployee(
                "Bob", "Lewis", "444-44-4444", 9, 2, 1965,8,24,2008, 5000, .03,
400 );
        BasePlusCommissionEmployee basePlusCommissionEmployee1 =
                new BasePlusCommissionEmployee(
                "Moon", "King", "263-34-2984", 10, 2, 1965,8,24,2009, 10000,
.04, 300 );

        System.out.println( "\nEmployees processed individually:\n" );

        System.out.printf( "%s\n%s: $%,.2f\n\n",
                salariedEmployee, "earned", salariedEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                salariedEmployee1, "earned", salariedEmployee1.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                salariedEmployee2, "earned", salariedEmployee2.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                hourlyEmployee, "earned", hourlyEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                hourlyEmployee1, "earned", hourlyEmployee1.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                hourlyEmployee2, "earned", hourlyEmployee2.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                commissionEmployee, "earned", commissionEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                commissionEmployee1, "earned", commissionEmployee1.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                basePlusCommissionEmployee,
                "earned", basePlusCommissionEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                basePlusCommissionEmployee1,
                "earned", basePlusCommissionEmployee1.earnings() );

        aBag.add(salariedEmployee);
        aBag.add(salariedEmployee1);
        aBag.add(salariedEmployee2);
        aBag.add(hourlyEmployee);
        aBag.add(hourlyEmployee1);
        aBag.add(hourlyEmployee2);
        aBag.add(commissionEmployee);
        aBag.add(commissionEmployee1);
        aBag.add(basePlusCommissionEmployee);
        aBag.add(basePlusCommissionEmployee1);

        Scanner input = new Scanner( System.in ); // to get current month
        int currentMonth;
        int currentYear;
        int currentDay;
        Date currentDate;

        // get and validate current month
        do
        {
            System.out.print("Enter the current year: ");
            currentYear = input.nextInt();
            System.out.print( "Enter the current month (1 - 12): " );
            currentMonth = input.nextInt();
            System.out.print("Enter the current day: ");
            currentDay = input.nextInt();
            System.out.println();
            currentDate = new Date(currentMonth, currentDay, currentYear);

         } while ( (( currentMonth < 1 ) || ( currentMonth > 12 )) ||
(currentDate.getYear() < 0) || (currentDate.getDay() == -1));

         System.out.printf("Current Date : %s\n\n",currentDate.toString());

         System.out.println( "Employees processed polymorphically:\n" );


         int lengthOfaBag = aBag.getCurrentSize();
         // generically process each element in array employees
         for (int index = 0; index < lengthOfaBag;index++)
         {
             Employee currentEmployee = aBag.remove();
             System.out.printf( "Employee %d is a %s\n", index,
                     currentEmployee.getClass().getName() );
             System.out.println( currentEmployee ); // invokes toString

             // determine whether element is a BasePlusCommissionEmployee
             if ( currentEmployee instanceof BasePlusCommissionEmployee )
             {
                 // downcast Employee reference to
                 // BasePlusCommissionEmployee reference
                 BasePlusCommissionEmployee employee =
                         ( BasePlusCommissionEmployee ) currentEmployee;
```

```
                   double oldBaseSalary = employee.getBaseSalary();
                   employee.setBaseSalary( 1.10 * oldBaseSalary );
                   System.out.printf(
                           "new base salary with 10%% increase is: $%,.2f\n",
                           employee.getBaseSalary() );
               } // end if

               // if month of employee's birthday, add $100 to salary
               // if year of hired date over 10, add 10% of salary
               if ( (currentYear - currentEmployee.getHiredDate().getYear()>10)||
                       ((currentYear - currentEmployee.getHiredDate().getYear() ==
10) && (currentMonth-currentEmployee.getHiredDate().getMonth() > 0))||
                       ((currentYear - currentEmployee.getHiredDate().getYear() ==
10) && (currentMonth-currentEmployee.getHiredDate().getMonth() == 0)&&
(currentDay-currentEmployee.getHiredDate().getDay()>=0)))
               {
                   if ( currentMonth == currentEmployee.getBirthDate().getMonth())
                       System.out.printf(
                               "earned $%,.2f %s %s\n\n",
((currentEmployee.earnings()*4)+100)*1.1,
                               "plus $100.00 birthday bonus","plus 10% of salary
bonus" );
                   else
                       System.out.printf(
                               "earned $%,.2f %s\n\n",
currentEmployee.earnings()*4*1.1,"plus 10% of salary bonus" );

               }else {

                   if ( currentMonth == currentEmployee.getBirthDate().getMonth())
                       System.out.printf(
                               "earned $%,.2f %s\n\n",
(currentEmployee.earnings()*4)+100,
                               "plus $100.00 birthday bonus");
                   else
                       System.out.printf(
                               "earned $%,.2f \n\n",
currentEmployee.earnings()*4);
               }


           } // end for

      } //end main
}
```

5) PayrollSystemTest_LinkedBag

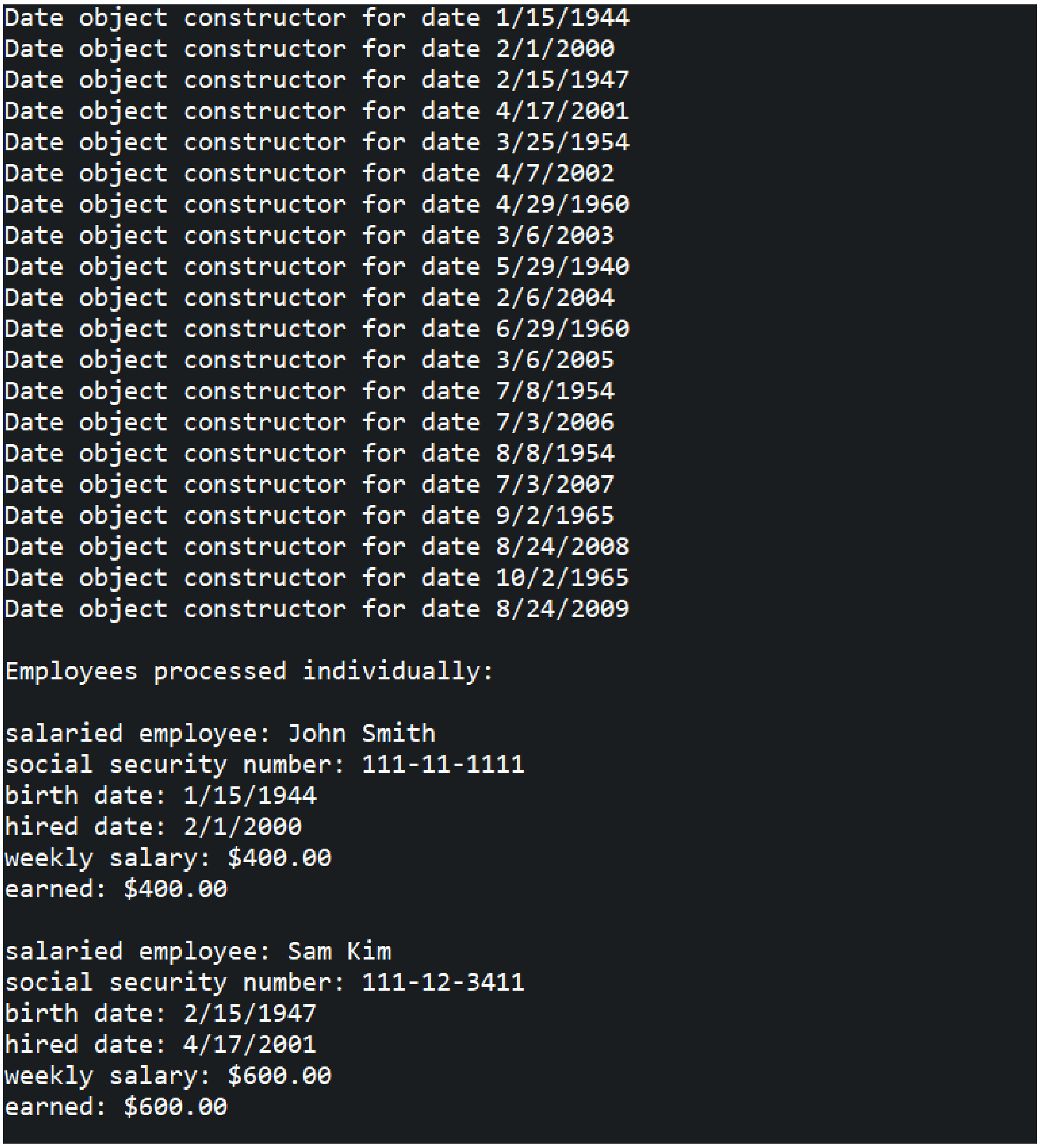
LinkedBag을 사용해 PayrollSystemTest 구현

```
// Employee hierarchy test program.
import java.util.Scanner; // program uses Scanner to obtain user input

public class PayrollSystemTest_LinkedBag
{
    public static void main( String[] args )
    {

        BagInterface<Employee> aBag = new LinkedBag<>();

        // create subclass objects
        SalariedEmployee salariedEmployee =
           new SalariedEmployee(
           "John", "Smith", "111-11-1111", 1, 15, 1944, 2,1,2000, 400.00 );
        SalariedEmployee salariedEmployee1 =
                new SalariedEmployee(
                "Sam", "Kim", "111-12-3411", 2, 15, 1947, 4,17,2001, 600.00 );
        SalariedEmployee salariedEmployee2 =
                new SalariedEmployee(
                "Dan", "David", "351-13-1251", 3, 25, 1954, 4,7,2002, 800.00 );
        HourlyEmployee hourlyEmployee =
                new HourlyEmployee(
                "Karen", "Price", "222-22-2222", 4, 29, 1960,3,6,2003 ,12.75, 40
);
        HourlyEmployee hourlyEmployee1 =
                new HourlyEmployee(
                "Lim", "Deep", "462-22-2522", 5, 29, 1940,2,6,2004 ,14.75, 40 );
        HourlyEmployee hourlyEmployee2 =
                new HourlyEmployee(
                "Koo", "Cold", "364-23-2352", 6, 29, 1960,3,6,2005 ,16.75, 40 );
        CommissionEmployee commissionEmployee =
                new CommissionEmployee(
                "Sue", "Jones", "333-33-3333", 7, 8, 1954,7,3,2006, 5000, .06 );
        CommissionEmployee commissionEmployee1 =
                new CommissionEmployee(
                "Soo", "Kim", "333-33-1113", 8, 8, 1954,7,3,2007, 10000, .06 );

        BasePlusCommissionEmployee basePlusCommissionEmployee =
                new BasePlusCommissionEmployee(
                "Bob", "Lewis", "444-44-4444", 9, 2, 1965,8,24,2008, 5000, .03,
400 );
        BasePlusCommissionEmployee basePlusCommissionEmployee1 =
                new BasePlusCommissionEmployee(
                "Moon", "King", "263-34-2984", 10, 2, 1965,8,24,2009, 10000,
.04, 300 );

        System.out.println( "\nEmployees processed individually:\n" );

        System.out.printf( "%s\n%s: $%,.2f\n\n",
                salariedEmployee, "earned", salariedEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                salariedEmployee1, "earned", salariedEmployee1.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                salariedEmployee2, "earned", salariedEmployee2.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                hourlyEmployee, "earned", hourlyEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                hourlyEmployee1, "earned", hourlyEmployee1.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                hourlyEmployee2, "earned", hourlyEmployee2.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                commissionEmployee, "earned", commissionEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                commissionEmployee1, "earned", commissionEmployee1.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                basePlusCommissionEmployee,
                "earned", basePlusCommissionEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n",
                basePlusCommissionEmployee1,
                "earned", basePlusCommissionEmployee1.earnings() );

        aBag.add(salariedEmployee);
        aBag.add(salariedEmployee1);
        aBag.add(salariedEmployee2);
        aBag.add(hourlyEmployee);
        aBag.add(hourlyEmployee1);
        aBag.add(hourlyEmployee2);
        aBag.add(commissionEmployee);
        aBag.add(commissionEmployee1);
        aBag.add(basePlusCommissionEmployee);
        aBag.add(basePlusCommissionEmployee1);

        Scanner input = new Scanner( System.in ); // to get current month
        int currentMonth;
        int currentYear;
        int currentDay;
        Date currentDate;

        // get and validate current month
        do
        {
            System.out.print("Enter the current year: ");
            currentYear = input.nextInt();
            System.out.print( "Enter the current month (1 - 12): " );
            currentMonth = input.nextInt();
            System.out.print("Enter the current day: ");
            currentDay = input.nextInt();
            System.out.println();
            currentDate = new Date(currentMonth, currentDay, currentYear);

         } while ( (( currentMonth < 1 ) || ( currentMonth > 12 )) ||
(currentDate.getYear() < 0) || (currentDate.getDay() == -1));

         System.out.printf("Current Date : %s\n\n",currentDate.toString());

         System.out.println( "Employees processed polymorphically:\n" );


         int lengthOfaBag = aBag.getCurrentSize();
         // generically process each element in array employees
         for (int index = 0; index < lengthOfaBag;index++)
         {
             Employee currentEmployee = aBag.remove();
             System.out.printf( "Employee %d is a %s\n", index,
                     currentEmployee.getClass().getName() );
             System.out.println( currentEmployee ); // invokes toString

             // determine whether element is a BasePlusCommissionEmployee
             if ( currentEmployee instanceof BasePlusCommissionEmployee )
             {
                 // downcast Employee reference to
                 // BasePlusCommissionEmployee reference
```

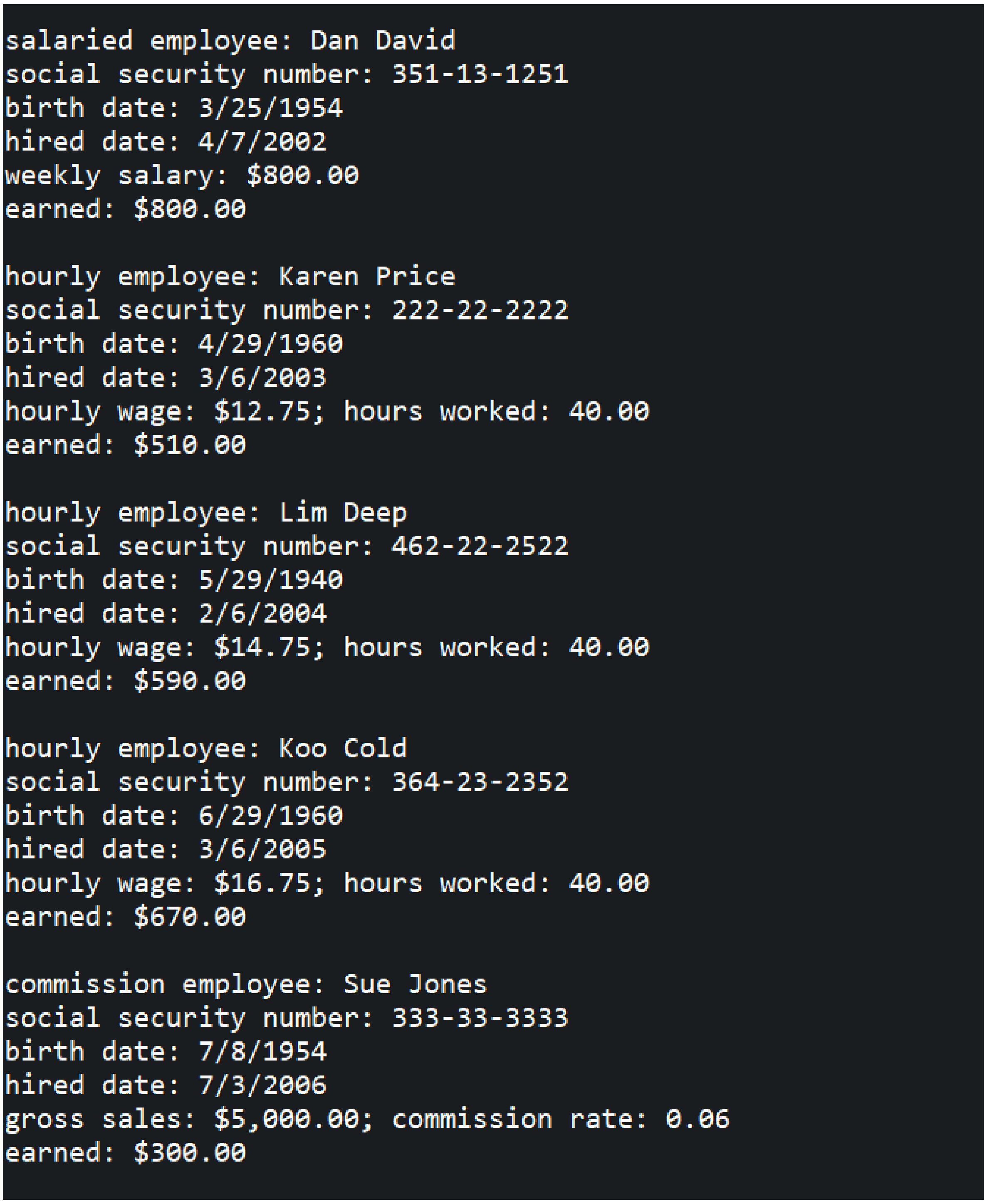
```
                   BasePlusCommissionEmployee employee =
                           ( BasePlusCommissionEmployee ) currentEmployee;

                   double oldBaseSalary = employee.getBaseSalary();
                   employee.setBaseSalary( 1.10 * oldBaseSalary );
                   System.out.printf(
                           "new base salary with 10%% increase is: $%,.2f\n",
                           employee.getBaseSalary() );
               } // end if

               // if month of employee's birthday, add $100 to salary
               // if year of hired date over 10, add 10% of salary
               if ( (currentYear - currentEmployee.getHiredDate().getYear()>10)||
                       ((currentYear - currentEmployee.getHiredDate().getYear() ==
10) && (currentMonth-currentEmployee.getHiredDate().getMonth() > 0))||
                       ((currentYear - currentEmployee.getHiredDate().getYear() ==
10) && (currentMonth-currentEmployee.getHiredDate().getMonth() == 0)&&
(currentDay-currentEmployee.getHiredDate().getDay()>=0)))
               {
                   if ( currentMonth == currentEmployee.getBirthDate().getMonth())
                       System.out.printf(
                               "earned $%,.2f %s %s\n\n",
((currentEmployee.earnings()*4)+100)*1.1,
                               "plus $100.00 birthday bonus","plus 10% of salary
bonus" );
                   else
                       System.out.printf(
                               "earned $%,.2f %s\n\n",
currentEmployee.earnings()*4*1.1,"plus 10% of salary bonus" );

               }else {

                   if ( currentMonth == currentEmployee.getBirthDate().getMonth())
                       System.out.printf(
                               "earned $%,.2f %s\n\n",
(currentEmployee.earnings()*4)+100,
                               "plus $100.00 birthday bonus");
                   else
                       System.out.printf(
                               "earned $%,.2f \n\n",
currentEmployee.earnings()*4);
               }


           } // end for

       } //end main
}
```

2. Screen Dump (실행 결과)

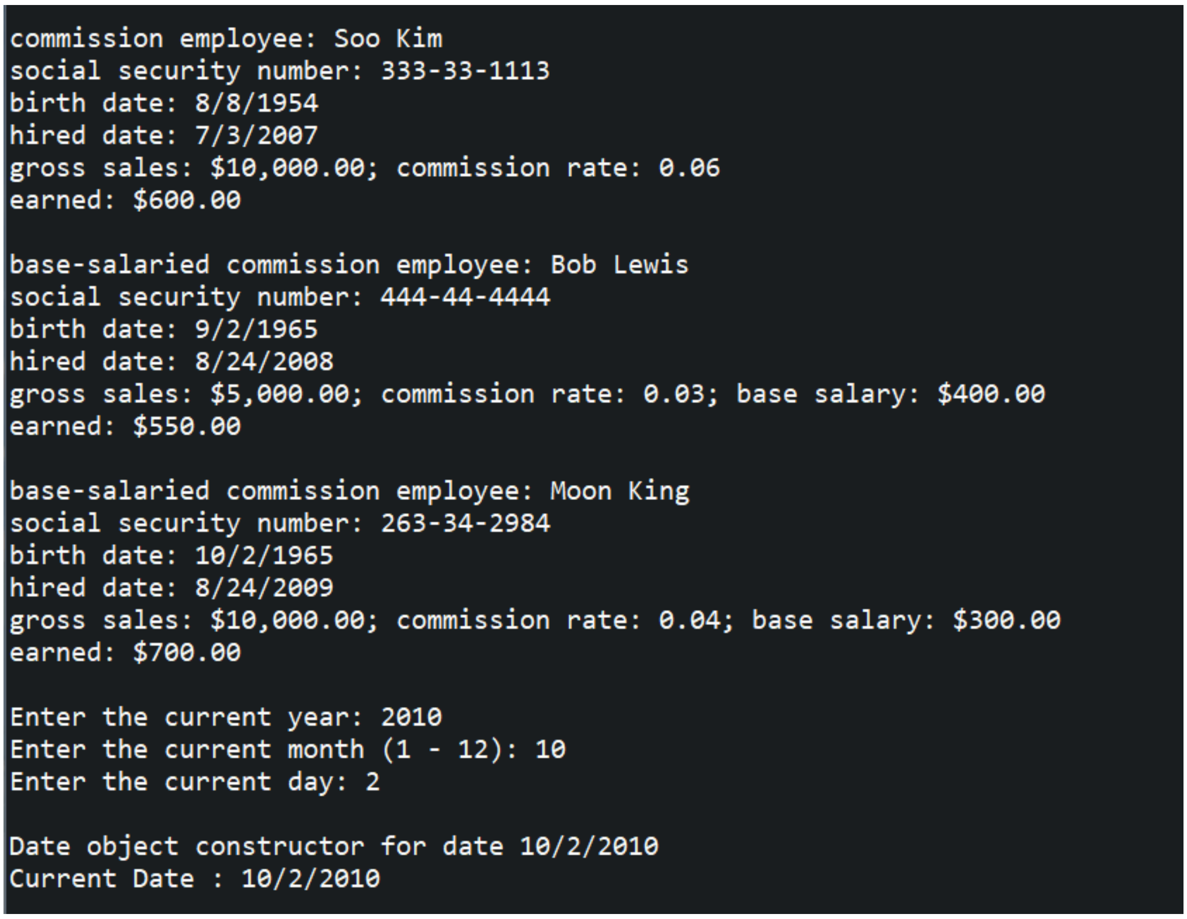
PayrollSystemTest_ArrayBag와 PayrollSystemTest_LinkedBag에서 동일하게 출력됨.



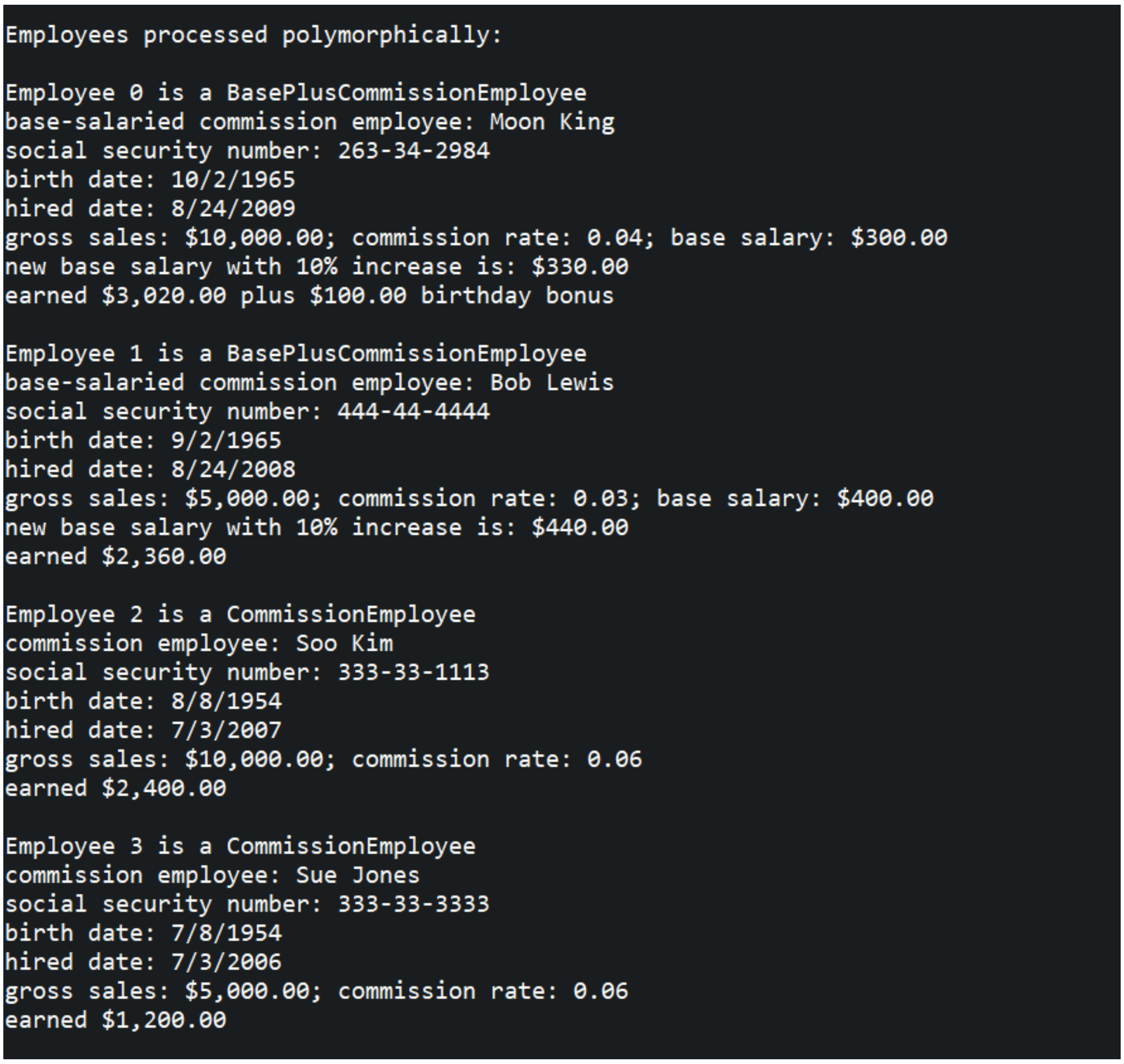
```
Date object constructor for date 1/15/1944
Date object constructor for date 2/1/2000
Date object constructor for date 2/15/1947
Date object constructor for date 4/17/2001
Date object constructor for date 3/25/1954
Date object constructor for date 4/7/2002
Date object constructor for date 4/29/1960
Date object constructor for date 3/6/2003
Date object constructor for date 5/29/1940
Date object constructor for date 2/6/2004
Date object constructor for date 6/29/1960
Date object constructor for date 3/6/2005
Date object constructor for date 7/8/1954
Date object constructor for date 7/3/2006
Date object constructor for date 8/8/1954
Date object constructor for date 7/3/2007
Date object constructor for date 9/2/1965
Date object constructor for date 8/24/2008
Date object constructor for date 10/2/1965
Date object constructor for date 8/24/2009

Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
birth date: 1/15/1944
hired date: 2/1/2000
weekly salary: $400.00
earned: $400.00

salaried employee: Sam Kim
social security number: 111-12-3411
birth date: 2/15/1947
hired date: 4/17/2001
weekly salary: $600.00
earned: $600.00
```



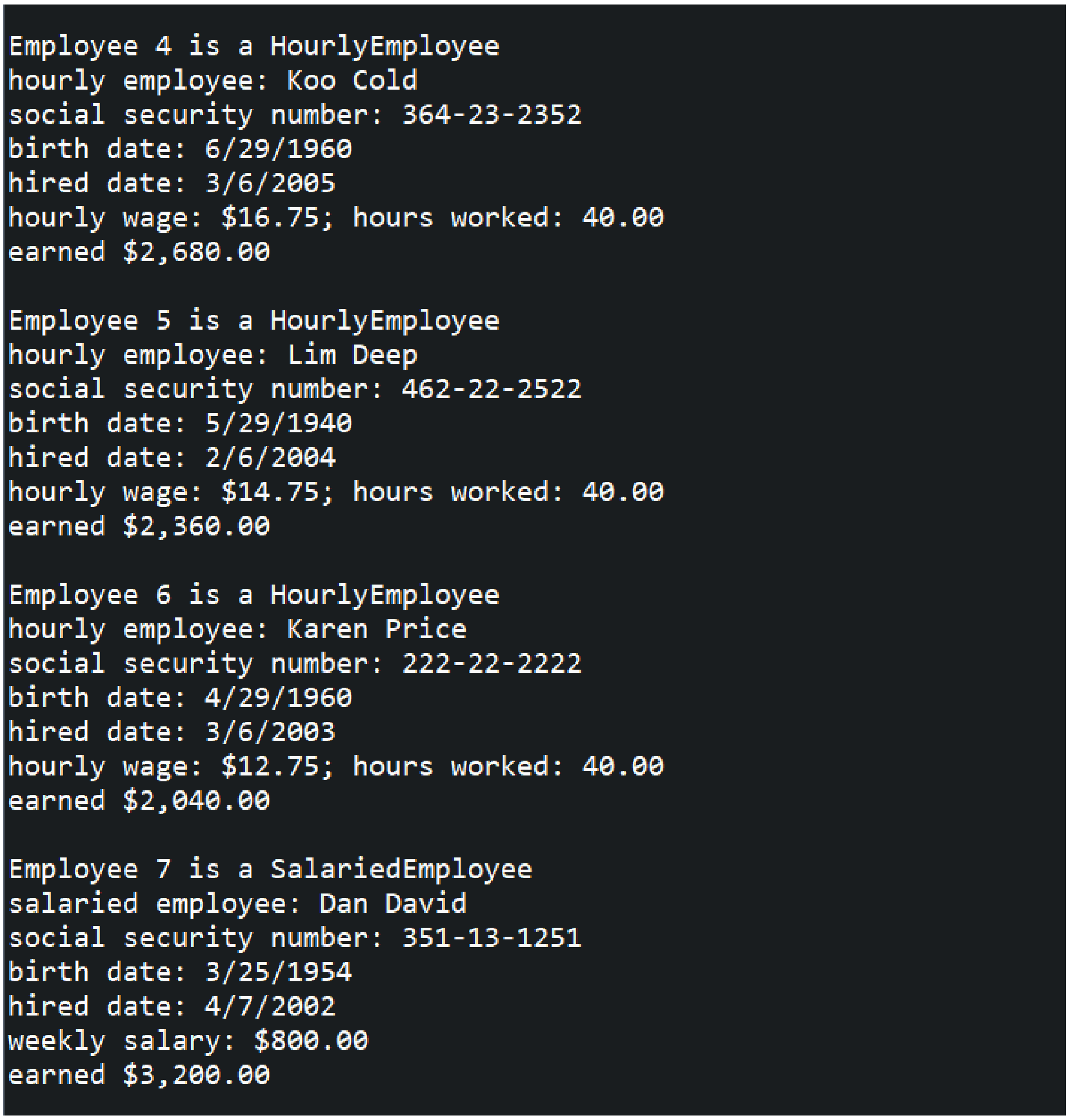
```
salaried employee: Dan David
social security number: 351-13-1251
birth date: 3/25/1954
hired date: 4/7/2002
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
birth date: 4/29/1960
hired date: 3/6/2003
hourly wage: $12.75; hours worked: 40.00
earned: $510.00

hourly employee: Lim Deep
social security number: 462-22-2522
birth date: 5/29/1940
hired date: 2/6/2004
hourly wage: $14.75; hours worked: 40.00
earned: $590.00

hourly employee: Koo Cold
social security number: 364-23-2352
birth date: 6/29/1960
hired date: 3/6/2005
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
birth date: 7/8/1954
hired date: 7/3/2006
gross sales: $5,000.00; commission rate: 0.06
earned: $300.00
```



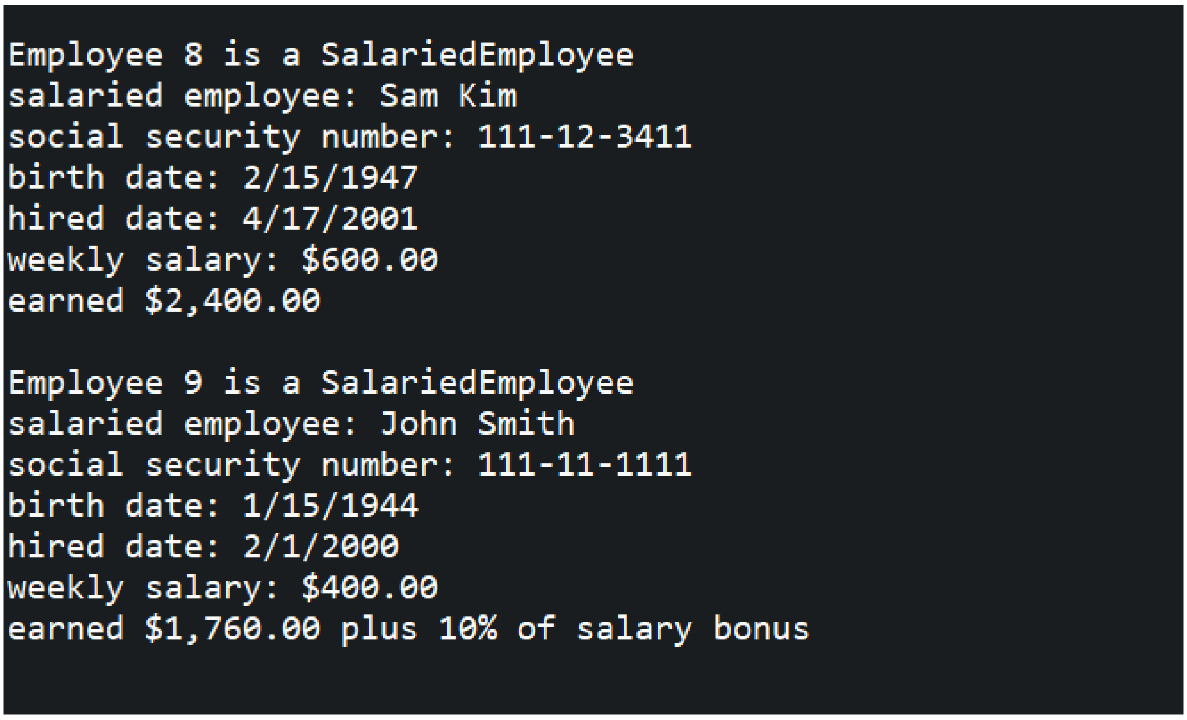
```
commission employee: Soo Kim
social security number: 333-33-1113
birth date: 8/8/1954
hired date: 7/3/2007
gross sales: $10,000.00; commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
birth date: 9/2/1965
hired date: 8/24/2008
gross sales: $5,000.00; commission rate: 0.03; base salary: $400.00
earned: $550.00

base-salaried commission employee: Moon King
social security number: 263-34-2984
birth date: 10/2/1965
hired date: 8/24/2009
gross sales: $10,000.00; commission rate: 0.04; base salary: $300.00
earned: $700.00

Enter the current year: 2010
Enter the current month (1 - 12): 10
Enter the current day: 2

Date object constructor for date 10/2/2010
Current Date : 10/2/2010
```



```
Employees processed polymorphically:

Employee 0 is a BasePlusCommissionEmployee
base-salaried commission employee: Moon King
social security number: 263-34-2984
birth date: 10/2/1965
hired date: 8/24/2009
gross sales: $10,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $3,020.00 plus $100.00 birthday bonus

Employee 1 is a BasePlusCommissionEmployee
base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
birth date: 9/2/1965
hired date: 8/24/2008
gross sales: $5,000.00; commission rate: 0.03; base salary: $400.00
new base salary with 10% increase is: $440.00
earned $2,360.00

Employee 2 is a CommissionEmployee
commission employee: Soo Kim
social security number: 333-33-1113
birth date: 8/8/1954
hired date: 7/3/2007
gross sales: $10,000.00; commission rate: 0.06
earned $2,400.00

Employee 3 is a CommissionEmployee
commission employee: Sue Jones
social security number: 333-33-3333
birth date: 7/8/1954
hired date: 7/3/2006
gross sales: $5,000.00; commission rate: 0.06
earned $1,200.00
```



```
Employee 4 is a HourlyEmployee
hourly employee: Koo Cold
social security number: 364-23-2352
birth date: 6/29/1960
hired date: 3/6/2005
hourly wage: $16.75; hours worked: 40.00
earned $2,680.00

Employee 5 is a HourlyEmployee
hourly employee: Lim Deep
social security number: 462-22-2522
birth date: 5/29/1940
hired date: 2/6/2004
hourly wage: $14.75; hours worked: 40.00
earned $2,360.00

Employee 6 is a HourlyEmployee
hourly employee: Karen Price
social security number: 222-22-2222
birth date: 4/29/1960
hired date: 3/6/2003
hourly wage: $12.75; hours worked: 40.00
earned $2,040.00

Employee 7 is a SalariedEmployee
salaried employee: Dan David
social security number: 351-13-1251
birth date: 3/25/1954
hired date: 4/7/2002
weekly salary: $800.00
earned $3,200.00
```



```
Employee 8 is a SalariedEmployee
salaried employee: Sam Kim
social security number: 111-12-3411
birth date: 2/15/1947
hired date: 4/17/2001
weekly salary: $600.00
earned $2,400.00

Employee 9 is a SalariedEmployee
salaried employee: John Smith
social security number: 111-11-1111
birth date: 1/15/1944
hired date: 2/1/2000
weekly salary: $400.00
earned $1,760.00 plus 10% of salary bonus
```

3. Javadoc 프로그램을 실행한 결과

1) BagInterface Javadoc

Interface BagInterface<T>

All Known Implementing Classes:
ArrayBag, LinkedBag

```
public interface BagInterface<T>
```

An interface that describes the operations of a bag of objects.

Author:
20171620 MoonSeongchan

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
boolean	add(T newEntry) Adds a new entry to this bag.
void	clear() Removes all entries from this bag
boolean	contains(T anEntry) Tests whether this bag contains a given entry.
int	getCurrentSize() Gets the current number of entries in this bag.
int	getFrequencyOf(T anEntry) Counts the number of times a given entry appears in this bag.
boolean	isEmpty() Sees whether this bag is empty.
T	remove() Removes one unspecified entry from this bag, if possible.
boolean	remove(T anEntry) Removes one occurrence of a given entry from this bag, if possible.
T[]	toArray() Retrieves all entries that are in this bag.

Method Detail

getCurrentSize

```
int getCurrentSize()
```

Gets the current number of entries in this bag.

Returns:
The integer number of entries currently in the bag

isEmpty

```
boolean isEmpty()
```

Sees whether this bag is empty.

Returns:
True if the bag is empty, or false if not.

add

```
boolean add(T newEntry)
```

Adds a new entry to this bag.

Parameters:
newEntry - The object to be added as a new entry.

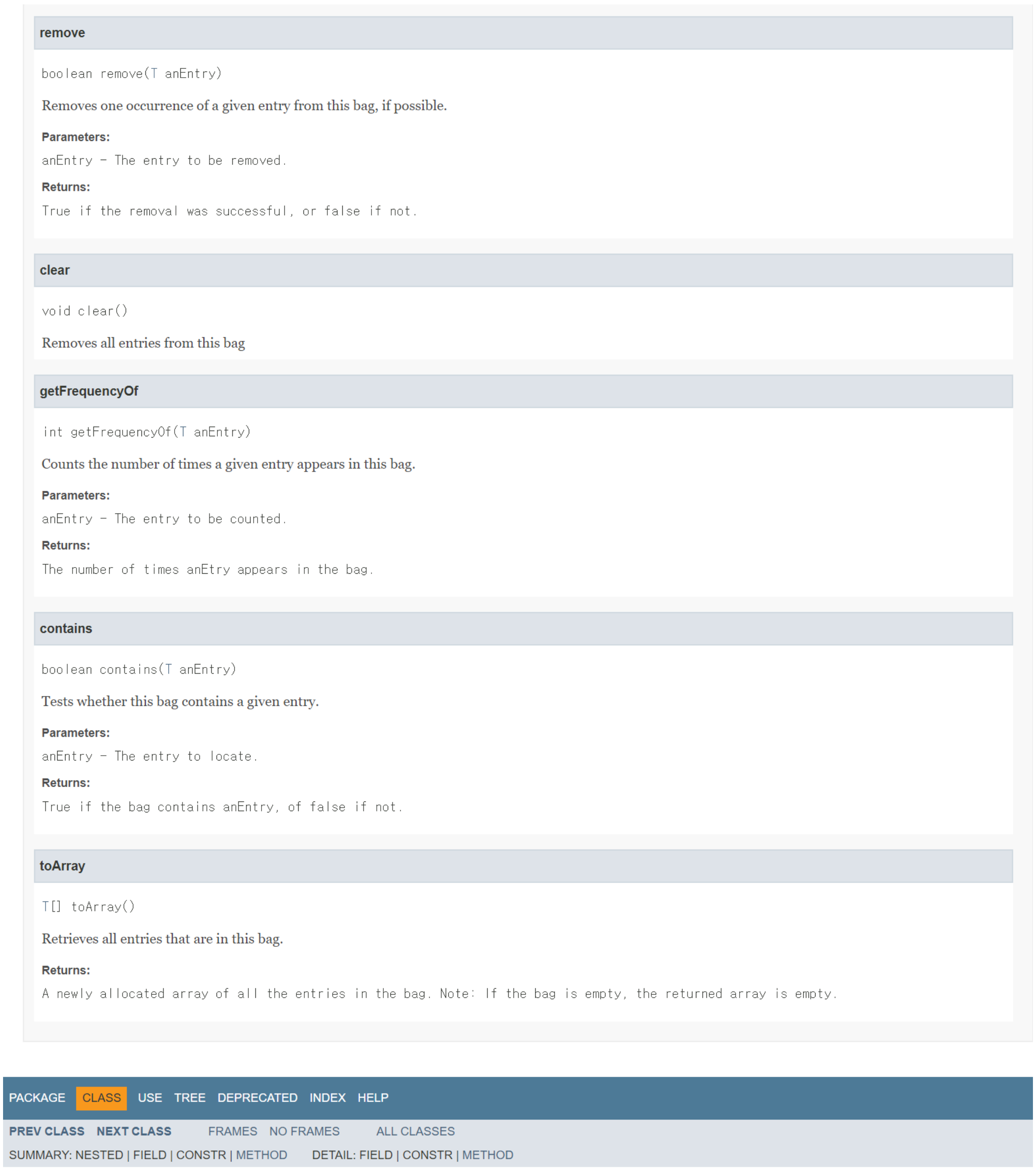
Returns:
True if the addition is successful, or false if not.

remove

```
T remove()
```

Removes one unspecified entry from this bag, if possible.

Returns:
Either the removed entry, if the removal was successful, or null



remove

```
boolean remove(T anEntry)
```

Removes one occurrence of a given entry from this bag, if possible.

Parameters:

`anEntry - The entry to be removed.`

Returns:

`True if the removal was successful, or false if not.`

clear

```
void clear()
```

Removes all entries from this bag

getFrequencyOf

```
int getFrequencyOf(T anEntry)
```

Counts the number of times a given entry appears in this bag.

Parameters:

`anEntry - The entry to be counted.`

Returns:

`The number of times anEtry appears in the bag.`

contains

```
boolean contains(T anEntry)
```

Tests whether this bag contains a given entry.

Parameters:

`anEntry - The entry to locate.`

Returns:

`True if the bag contains anEntry, of false if not.`

toArray

```
T[] toArray()
```

Retrieves all entries that are in this bag.

Returns:

`A newly allocated array of all the entries in the bag. Note: If the bag is empty, the returned array is empty.`

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

2) ArrayBag Javadoc

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class ArrayBag<T>

java.lang.Object
    ArrayBag<T>

All Implemented Interfaces:

`BagInterface<T>`

```
public final class ArrayBag<T>
extends java.lang.Object
implements BagInterface<T>
```

A class of bags whose entries are stored in a fixed-size array.

Author:

`20171620 MoonSeongchan`

Constructor Summary

Constructors

Constructor and Description
`ArrayBag()` Creates an empty bag whose capacity is 25.
`ArrayBag(int desiredCapacity)` Creates an empty bag having a given capacity

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`add(T newEntry)` Adds a new entry to this bag.
`void`	`clear()` Removes all entries from this bag.
`boolean`	`contains(T anEntry)` Check if there is anEntry in this bag.
`int`	`getCurrentSize()` Returns current size of bag
`int`	`getFrequencyOf(T anEntry)` Counts the number of times a given entry appears in this bag.
`boolean`	`isEmpty()` Returns True if the bag is empty, or false if not.
`T`	`remove()` Remove one unspecified entry from this bag, if possible
`boolean`	`remove(T anEntry)` Removes one occurrence of a given entry from this bag, if possible.
`T[]`	`toArray()` Retrieves all entries that are in this bag.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

ArrayBag

`public ArrayBag()`

Creates an empty bag whose capacity is 25.

ArrayBag

`public ArrayBag(int desiredCapacity)`

Creates an empty bag having a given capacity

Parameters:

`desiredCapacity - The integer capacity desired.`

Method Detail

add

`public boolean add(T newEntry)`

Adds a new entry to this bag.

Specified by:
`add` in interface `BagInterface<T>`

Parameters:
`newEntry` - The object to be added as a new entry.

Returns:
True.

toArray

`public T[] toArray()`

Retrieves all entries that are in this bag.

Specified by:
`toArray` in interface `BagInterface<T>`

Returns:
A newly allocated array of all the entries in the bag.

isEmpty

`public boolean isEmpty()`

Returns True if the bag is empty, or false if not.

Specified by:
`isEmpty` in interface `BagInterface<T>`

Returns:
True if the bag is empty, or false if not.

getCurrentSize

`public int getCurrentSize()`

Returns current size of bag

Specified by:
`getCurrentSize` in interface `BagInterface<T>`

Returns:
The integer number of entries currently in the bag

getFrequencyOf

`public int getFrequencyOf(T anEntry)`

Counts the number of times a given entry appears in this bag.

Specified by:
`getFrequencyOf` in interface `BagInterface<T>`

Parameters:
`anEntry` - The entry to be counted.

Returns:
The number of times anEntry appears in the bag.

contains

`public boolean contains(T anEntry)`

Check if there is anEntry in this bag.

Specified by:
`contains` in interface `BagInterface<T>`

Parameters:
`anEntry` - The entry to be searched.

Returns:
boolean value the result of search.

clear

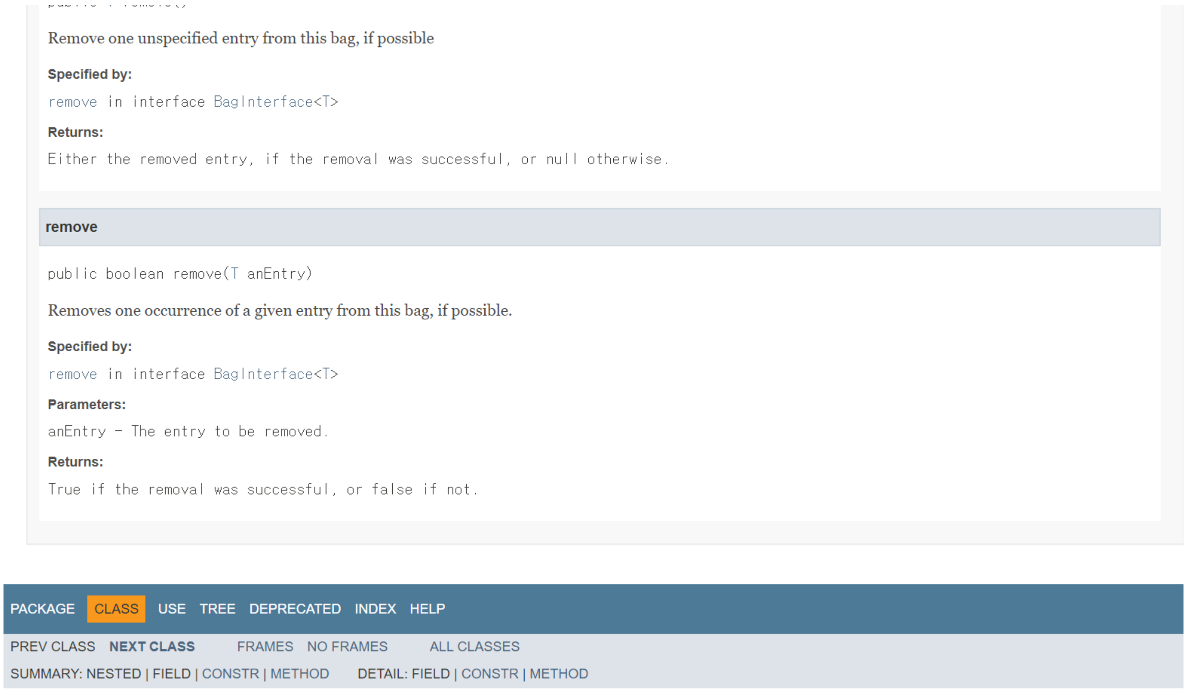
`public void clear()`

Removes all entries from this bag.

Specified by:
`clear` in interface `BagInterface<T>`

remove

`public T remove()`



Remove one unspecified entry from this bag, if possible

Specified by:

`remove in interface BagInterface<T>`

Returns:

`Either the removed entry, if the removal was successful, or null otherwise.`

remove

```
public boolean remove(T anEntry)
```

Removes one occurrence of a given entry from this bag, if possible.

Specified by:

`remove in interface BagInterface<T>`

Parameters:

`anEntry - The entry to be removed.`

Returns:

`True if the removal was successful, or false if not.`

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

3) LinkedBag Javadoc

Class LinkedBag<T>

java.lang.Object
    LinkedBag<T>

All Implemented Interfaces:

BagInterface<T>

```
public final class LinkedBag<T>
extends java.lang.Object
implements BagInterface<T>
```

A class of bags whose entries are stored in a chain of linked nodes. The bag is never full.

Author:

20171620 MoonSeongchan

Constructor Summary

Constructors

Constructor and Description
`LinkedBag()` Creates an empty bag whose firstNode is null.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`add(T newEntry)` Adds a new entry to this bag.
`void`	`clear()` Removes all entries from this bag.
`boolean`	`contains(T anEntry)` Check if there is anEntry in this bag.
`int`	`getCurrentSize()` Returns current size of bag
`int`	`getFrequencyOf(T anEntry)` Counts the number of times a given entry appears in this bag.
`boolean`	`isEmpty()` Returns True if the bag is empty, or false if not.
`T`	`remove()` Remove one unspecified entry from this bag, if possible
`boolean`	`remove(T anEntry)` Removes one occurrence of a given entry from this bag, if possible.
`T[]`	`toArray()` Retrieves all entries that are in this bag.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

LinkedBag

```
public LinkedBag()
```

Creates an empty bag whose firstNode is null.

Method Detail

add

```
public boolean add(T newEntry)
```

Adds a new entry to this bag.

Specified by:

`add` in interface `BagInterface<T>`

Parameters:

`newEntry` - The object to be added as a new entry.

Returns:

True.

toArray

`public T[] toArray()`

Retrieves all entries that are in this bag.

Specified by:

`toArray in interface BagInterface<T>`

Returns:

A newly allocated array of all the entries in the bag.

getFrequencyOf

`public int getFrequencyOf(T anEntry)`

Counts the number of times a given entry appears in this bag.

Specified by:

`getFrequencyOf in interface BagInterface<T>`

Parameters:

`anEntry` - The entry to be counted.

Returns:

The number of times anEntry appears in the bag.

contains

`public boolean contains(T anEntry)`

Check if there is anEntry in this bag.

Specified by:

`contains in interface BagInterface<T>`

Parameters:

`anEntry` - The entry to be searched.

Returns:

boolean value the result of search.

remove

`public T remove()`

Remove one unspecified entry from this bag, if possible

Specified by:

`remove in interface BagInterface<T>`

Returns:

Either the removed object, if the removal was successful, or null.

remove

`public boolean remove(T anEntry)`

Removes one occurrence of a given entry from this bag, if possible.

Specified by:

`remove in interface BagInterface<T>`

Parameters:

`anEntry` - The entry to be removed.

Returns:

True if the removal was successful, or false otherwise.

isEmpty

`public boolean isEmpty()`

Returns True if the bag is empty, or false if not.

Specified by:

`isEmpty in interface BagInterface<T>`

Returns:

True if the bag is empty, or false if not.

getCurrentSize

`public int getCurrentSize()`

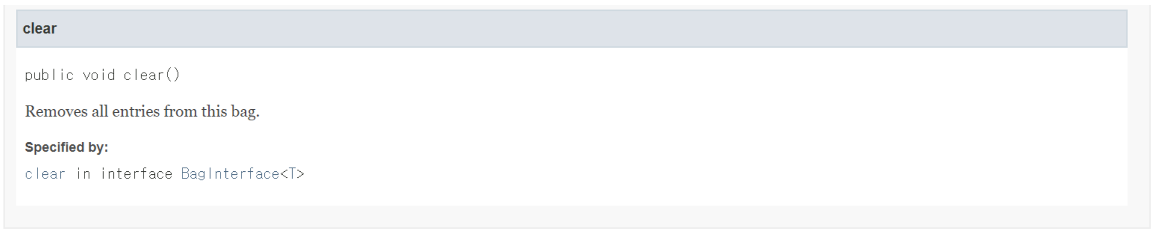
Returns current size of bag

Specified by:

`getCurrentSize in interface BagInterface<T>`

Returns:

The integer number of entries currently in the bag



clear

```
public void clear()
```

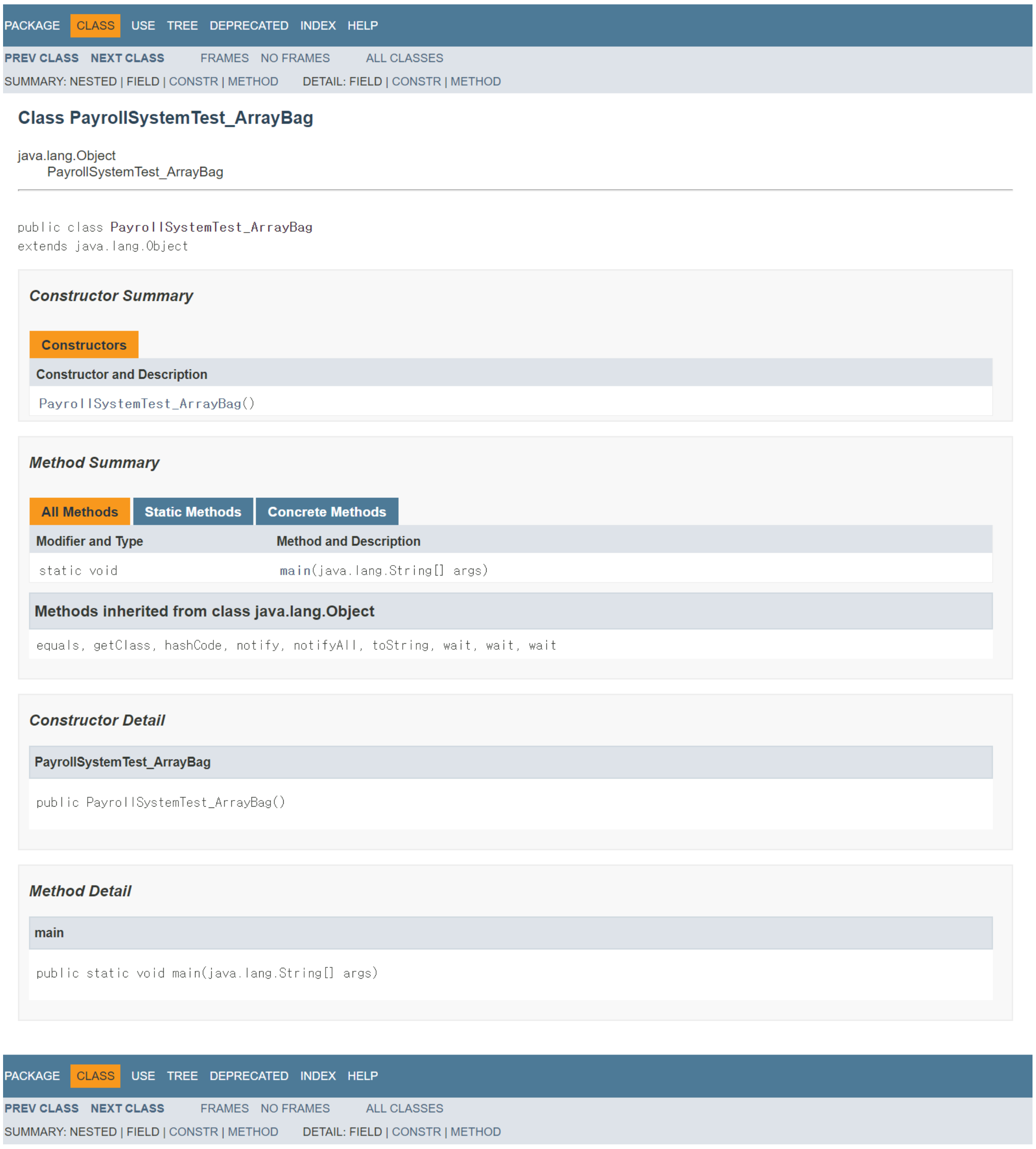
Removes all entries from this bag.

Specified by:

`clear in interface BagInterface<T>`

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

4) PayrollSystemTest_ArrayBag Javadoc



PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class PayrollSystemTest_ArrayBag

java.lang.Object
    PayrollSystemTest_ArrayBag

```
public class PayrollSystemTest_ArrayBag
extends java.lang.Object
```

Constructor Summary

Constructors

Constructor and Description
`PayrollSystemTest_ArrayBag()`

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
`static void`	`main(java.lang.String[] args)`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

PayrollSystemTest_ArrayBag

```
public PayrollSystemTest_ArrayBag()
```

Method Detail

main

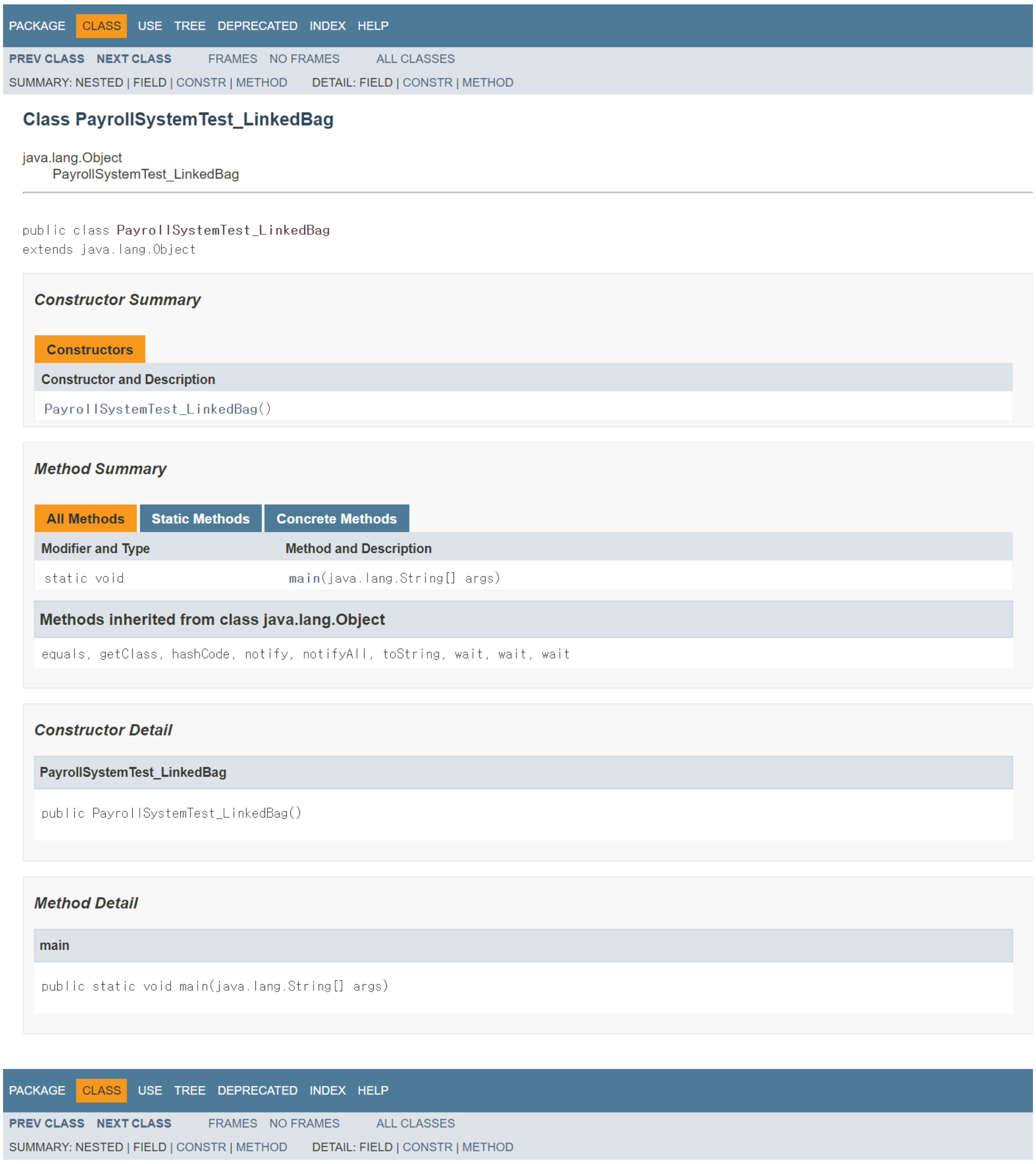
```
public static void main(java.lang.String[] args)
```

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5) PayrollSystemTest_LinkedBag Javadoc


PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class PayrollSystemTest_LinkedBag

java.lang.Object
    PayrollSystemTest_LinkedBag

```
public class PayrollSystemTest_LinkedBag
extends java.lang.Object
```

Constructor Summary

Constructors

Constructor and Description
`PayrollSystemTest_LinkedBag()`

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
`static void`	`main(java.lang.String[] args)`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

PayrollSystemTest_LinkedBag

```
public PayrollSystemTest_LinkedBag()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

4. Bag라는 자료구조를 사용하는 이유

집합 자료형(Set)는 원소들의 순서에 상관이 없지만 중복된 원소들을 가질 수 없습니다.
배열(Array)과 리스트(List)는 중복된 원소들을 가질 수 있지만 순서에 상관이 있는 자료형입니다.
Bag는 순서도 상관없고 중복된 원소들도 포함할 수 있는 자료형입니다.
이전에 PayrollSystemTest에 사용했던 배열은 크기를 처음에 지정해준 다음 원소들을 포함해 주었는데,
포함하려는 원소들이 해당 크기를 초과할 경우 다시 배열의 크기를 수정해주고
또 일일이 인덱스를 통해 추가해주어야 했습니다.
하지만 Bag 자료구조를 사용할 경우 포함되는 원소들의 갯수에 상관없이 추가할 수 있으며
일일이 크기를 수정해주는 작업을 할 필요가 없어집니다.
또한 Bag 인터페이스에 선언된 다양한 메서드들을 통해 Bag내에 원소 제거하기,
Bag의 크기 알아내기, 모든 원소들 불러오기 등 다양한 기능들을 메서드 호출을 통해 쉽게 사용할 수 있습니다.

5. 기타 제출물

- 소스 파일
- Data Structure Assignment #2 보고서 (pdf 파일)
- Data Structure Assignment #2 보고서 (html 파일) - #pdf 파일 출력 오류 대비