Rolf Axelsson & Jonas Petersson

Laboration 3 - Sökning

På laborationen används en del filer. Det är versionerna i kod_labb3.zip som ska användas! Packa upp filerna i en katalog med namnet **laboration3**. Se till att laboration3-katalogen är i projektets src-katalog.

Grundläggande uppgifter

I uppgifterna 1-7 ska du använda *linjär sökning.*

1. Skriv en metod som söker efter efter ett värde i en double-array:
   ```
   public int indexOf( double[] array, double value) {...}
   ```
   Metoden ska returnera positionen för det sökta värdet. Om det sökta värdet inte finns i arrayen ska metoden returnera -1.
   För att testa metoden kan du skapa en array med lite olika double-värden och sedan testsöka i arrayen.

2. Skriv en metod som söker efter en sträng i en array med strängar:
   ```
   public int indexOf( String[] array, String value) {...}
   ```
   Metoden ska returnera -1 om strängen inte finns i arrayen.

3. Utgå från klassen **Laboration3** (se kod_labb3.zip). Klassen innehåller metoden
   ```
   public int[] randomIntArray( int count ) {...}
   ```
   vilken skapar en array med count st element. Elementen har värdena 0 till count-1 och värdena är i slumpmässig ordning.

   Skriv ett program vilket skapar en array med 10000 element (ska anropa ovanstående metod) och som därefter kontrollerar tiden det tar att söka efter samtliga element som finns i arrayen. För att söka i arrayen kan du använda **indexOf**-metoden i klassen **LinearSearch1** (se kod_labb3.zip).

   Halvkod
   Skapa en array med 10000 element
   Kontrollera tiden just nu (long start = System.currenttimeMillis();)
   För varje tal 0-9999
      Sök efter talet (dvs. först 0, sedan 1, sedan 2, osv) i arrayen
   Kontrollera tiden på nytt (long stop = ...;)
   Skriv ut antalet millisekunder som sökningarna tog (stop – start).

4. Skriv klassmetoden **shuffle** i klassen **Exercise4**
   ```
   public static void shuffle( Object[] obj )
   ```
   vilken blandar elementen i en Object-array. Tänk på att även swap-metoden måste vara en klassmetod. Testa metoden med:

   ```
   Integer[] arr = new Integer[5];
   for( int i=0; i<arr.length; i++ ) {
       arr[i] = new Integer(i);
   }
   Exercise4.shuffle( arr );
   for( Integer elem : arr )
       System.out.println(elem);
   ```

   Till din hjälp har du klassen **Shuffle** i föreläsningsmaterialet.

5. Skriv metoden, **public RealNbr[] get RealNbrArray(int n)**. Metoden ska:
   * skapa en RealNbr -array med plats för n st element
   * fylla arrayen med RealNbr -objekt (se kod_labb3.zip)
   * blanda elementen i arrayen (anropa Exercise4.shuffle)
   * returnera arrayen

6. Skriv en metod som söker efter ett **RealNbr**-objekt i en array:
   ```
   public int indexOf( RealNbr[] array, RealNbr value) {...}
   ```
   Metoden ska returnera -1 om RealNbr-värdet inte finns i arrayen.

7. Skriv en metod som söker efter ett objekt i en Object-array. Sökningen ska ske med hjälp av equals-metoden.
   ```
   public int indexOf( Object[] array, Object obj) {...}
   ```
   Metoden ska returnera -1 om *obj* inte finns i arrayen.

   Test metoden med hjälp av
   * sökning av String-objekt i en String-array.
   * sökning av RealNbr-objekt i en RealNbr-array.

8. Metoden **fillInteger** i klassen **Laboration3** skapar och fyller en ArrayList med n st Integer-objekt, samtliga med slumpvärden i intervallet min till max:

```
public ArrayList<Integer> fillInteger( int n, int min, int max ) {
    int random;
    ArrayList<Integer> list = new ArrayList<Integer> ();
    for( int i = 0; i < n; i++ ) {
        random = ( int )( Math.random() * ( max - min + 1 ) ) + min;
        list.add( new Integer( random ) );
    }
    return list;
}
```

   Använd metoden för att få en ArrayList fylld med 100000 Integer-objekt i intervallet 10000 – 50000. Skapa sedan en ArrayList< RealNbr> och fyll listan med 100000 RealNbr-objekt. RealNbr-objekten ska ha samma värden som Integer-objekten i den första listan.

   Sortera listan med RealNbr-objekten med metoden Collection.sort(List list);. Slutligen ska du undersöka om talen 10000, 20000, 30000, 40000 och 50000 finns lagrade i ArrayList-objektet. Använd metoden
   ```
   Collections.binarySearch( List list, Object key);
   ```
   vid sökningarna. Det innebär att du måste skapa ett RealNbr-objekt att skicka med vid sökningen, t.ex.
   ```
   ... Collections.binarySearch( realNbrs, new RealNbr(10000) );
   ```

9. Gå till väga som i uppgift 3 men
   * sortera arrayen före sökningarna
   * använd **binarySearch**-metoden i klassen **BinarySearch**.

   Hur mycket snabbare går binär sökning jämfört med linjär sökning?

10. Skriv en **binarySearch**-metod vilken söker ett värde i en long-array:
    ```
    public int binarySearch( long[] array, long value) {...}
    ```
    Metoden ska använda en while-loop (se föreläsning). Metoden ska *inte* använda metoden Arrays.binarySearch för att lösa uppgiften.

11. Skriv en **binarySearch**-metod
    ```
    public int binarySearch( String[] array, String value )
    ```
    vilken söker en sträng i en array med String-objekt. För att jämföra strängar ska du använda *compareTo*-metoden i klassen String. Du ska *inte* använda metoden Collections.binarySearch för att lösa uppgiften.

Fördjupande uppgifter

12. Skriv en **binarySearch**-metod vilken söker ett objekt i en Objekt-array. Följande ska metoden uppfylla:
    * Metoden ska ha parameterlistan (Object[] array, Object key)
    * Metoden utgår från att objekten i arrayen implementerar Comparable. Det innebär att jämförelsen ungefär går till så här:
    ```
    Comparable comp = ( Comparable )key;
    :
    compare = key.compareTo( array[ pos ] );
    ```

13. Skriv en **binarySearch**-metod som söker efter objekt i ett List-objekt (t.ex. ArrayList, LinkedList,...). Följande ska metoden uppfylla:
    * Metoden ska ha parameterlistan (List list, Object key)
    * Metoden utgår från att objekten i List-objektet implementerar Comparable. Det innebär att jämförelsen ungefär går till så här:
    ```
    Comparable comp = ( Comparable )key;
    :
    compare = key.compareTo( list.get( i ) );
    ```

14. Klassen **Person(firstName, familyName, age)** är given (se kod_labb3.zip). Skriv en klass, **FamilyName**, vilken implementerar **Comparator<Person>** på så sätt att **Person**-objekt sorteras i bokstavsordning avseende efternamn.

15. Skriv en klass,**FamilyNameFirstName**, vilken implementerar **Comparator<Person>** på så sätt att **Person**-objekt sorteras i bokstavsordning avseende efternamn och vid lika efternamn i bokstavsordning avseende förnamn.

16. Skriv metoden **binarySearch(Object[] array, Object value, Comparator comp)**. Testa metoden genom att placera ett antal Person-objekt i en array (ej med identiskt för- och efternamn men en del med samma efternamn) och gör sedan sökningar. Använd först klassen FamilyName vid sökningarna och sedan klassen FamilyNameFirstName. Kontrollera noga resultatet. De kan i vissa fall bli felaktiga.

Extrauppgifter

17. Skriv metoden getArray, vilken ska skapa och fylla en Object-array (Object[]) med n st objekt. Objekten som ska fylla arrayen fås genom anrop till en ObjectCreator-implementering.

```
public Object[] getArray( int n, ObjectCreator creator) {...}
```

```
public interface ObjectCreator {
    public Object nextObject();
}
```

Exempel på ObjectCreator-implementering:

```
class RandomNumbers implements ObjectCreator {
    private int min, max;

    public RandomNumbers( int min, int max ) {
        this.min = min;
        this.max = max;
    }

    public Object nextObject() {
        int random = (int)( Math.random()*( max - min + 1 ) )+min;
        return new Integer(random);
    }
}
```

18. I klassen **Laboration3** hittar du en metod som heter *permute*. Den ska anropas med en sträng som argument. Metoden kommer att generera alla strängar som tecknen i den första strängen kan bygga upp, dock endast de där alla tecknen används. Strängarna returneras i en ArrayList<String>.

Exempel:

```
Laboration3 lab3 = new Laboration3();
ArrayList<String> list = lab3.permute( "OLA" );
System.out.println(list);
```

list kommer efter anropet att innehålla objekten: "OLA", "OAL", "LOA", "LAO", "AOL" och "ALO"

Antalet objekt växer snabbt: 3 tecken ger 6 objekt, 4 tecken ger 24 objekt, 5 tecken ger 120 objekt, 6 tecken ger 720 objekt. Du känner kanske igen tillväxten – det är samma som fakultet.

Skriv en implementering av ObjectCreator, ***Permutation***, som
* tar en sträng som argument till konstruktorn
* anropar *permute* med denna sträng
* returnerar dessa objekt (String-objekt) vid anrop till *nextObject*-metoden. Returnera objektet i position 0 första gången, position 1 nästa gång osv.. När det inte finns fler objekt att returnera så börja från början igen.

Testkod

```
Object[] perm = getArray( 8, new Permutation( "OLA" ) );
for( int i = 0; i < perm.length; i++ )
    system.out.println( perm[ i ] );
```

Lösningar

Uppgift 1

```
public int indexOf( double[] array, double value ) {
    for( int i = 0;  i<array.length; i++  ) {
        if( value == array[ i ] )
            return i;
    }
    return -1;
}
```

Uppgift 2

```
public int indexOf( String[] array, String value ) {
    for( int i = 0;  i<array.length; i++  ) {
        if( value.equals( array[ i ] ) )
            return i;
    }
    return -1;
}
```

Uppgift 3

```
public void uppgift3() {
    Laboration3 lab3 = new Laboration3();
    LinearSearch ls = new LinearSearch();
    int[] arr = lab3.randomIntArray(10000);
    int res;
    long stop, start = System.currentTimeMillis();
    for( int i=0; i<=9999; i++ ) {
        res = ls.indexOf(arr, i);
    }
    stop = System.currentTimeMillis();
    System.out.println("Tid: " + (stop-start) + " ms");
}
```

Uppgift 4

```
private static void swap( Object[] array, int elem1, int elem2 ) {
    Object temp = array[ elem1 ];
    array[ elem1 ] = array[ elem2 ];
    array[ elem2 ] = temp;
}

public static void shuffle( Object[] array ) {
    int pos;
    for( int i = array.length - 1; i > 0; i-- ) {
        pos = ( int )( Math.random() * ( i + 1 ) );
        swap( array, i, pos );
    }
}
```

Uppgift 5

```
public RealNbr[] getRealNbrArray( int n ) {
    RealNbr[] tal = new RealNbr[n];
    for( int i=0; i<tal.length; i++ )
        tal[i] = new RealNbr(i);
    Solution.shuffle(tal);
    return tal;
}
```

Uppgift 6

```
public int indexOf( RealNbr[] array, RealNbr value) {
    for( int i = 0;  i<array.length; i++  ) {
        if( value.equals( array[ i ] ) )
            return i;
    }
    return -1;
}
```

Uppgift 7

```
public int indexOf( Object[] array, Object obj) {
    for( int i = 0;  i<array.length; i++  ) {
        if( obj.equals( array[ i ] ) )
            return i;
    }
    return -1;
}
```

Uppgift 8

```
public void uppgift8() {
    Laboration3 lab3 = new Laboration3();
    ArrayList<Integer> arr1 = lab3.fillInteger( 100000, 10000, 50000 );
    ArrayList<RealNbr> arr2 = new ArrayList<RealNbr>();
    for(Integer i : arr1) {  // eller for(int i=0; i<arr1.size(); i++) { ...arr1.get(i).intValue() ...}
        arr2.add( new RealNbr( i.intValue() ) );
    }
    Collections.sort(arr2);
    for( int i=10000; i<=50000; i+=10000 ) {
        System.out.println( i + ": " + Collections.binarySearch(arr2, new
RealNbr(i)) );
    }
}
```

Uppgift 9

```
public void uppgift9() {
    Laboration3 lab3 = new Laboration3();
    BinarySearch bs = new BinarySearch();
    int[] arr = lab3.randomIntArray(10000);
    Arrays.sort(arr);
    int res;
    long stop, start = System.currentTimeMillis();
    for( int i=0; i<=9999; i++ ) {
        res = bs.binarySearch(arr, i);
        System.out.println(res);
    }
    stop = System.currentTimeMillis();
    System.out.println("Tid: " + (stop-start) + " ms");
}
```

Uppgift 10

```
public int binarySearch( long[] array , long value ) {
    int res = -1, min = 0, max = array.length - 1, pos;
    while( ( min <= max ) && ( res == -1 ) ) {
        pos = (min + max) / 2;
        if( value == array[ pos ] )
            res = pos;
        else if( value < array[ pos ])
            max = pos - 1;
        else
            min = pos + 1;
    }
    return res;
}
```

Uppgift 11

```
public int binarySearch( String[] array , String value ) {
    int res = -1, compare , min = 0, max = array.length - 1, pos;
    while( ( min <= max ) && ( res == -1 ) ) {
        pos = (min + max) / 2;
        compare = value.compareTo( array[ pos ] );
        if( compare == 0 )
            res = pos;
        else if( compare < 0 )
            max = pos - 1;
        else
            min = pos + 1;
    }
    return res;
}
```

Uppgift 12

```
public int binarySearch( Object[] array, Object key ) {
    int res = -1, compare, min = 0, max = array.length - 1, pos;
    Comparable comp = (Comparable)key;
    while( ( min <= max ) && ( res == -1 ) ) {
        pos = (min + max) / 2;
        compare = comp.compareTo( array[ pos ] );
        if( compare == 0 )
            res = pos;
        else if( compare < 0 )
            max = pos - 1;
        else
            min = pos + 1;
    }
    return res;
}
```

Uppgift 13

```
public int binarySearch( List list, Object key ) {
    int res = -1, compare, min = 0, max = list.size() - 1, pos;
    Comparable comp = (Comparable)key;
    while( ( min <= max ) && ( res == -1 ) ) {
        pos = (min + max) / 2;
        compare = comp.compareTo( list.get( pos ) );
        if( compare == 0 )
            res = pos;
        else if( compare < 0 )
            max = pos - 1;
        else
            min = pos + 1;
    }
    return res;
}
```

Uppgift 14

```
public class FamilyName implements Comparator<Person> {
    public int compare(Person p1, Person p2) {
        String fName1 = p1.getFamilyName();
        String fName2 = p2.getFamilyName();
        return fName1.compareTo(fName2);
    }
}
```

Uppgift 15

```
public class FamilyNameFirstName implements Comparator<Person> {
    public int compare(Person p1, Person p2) {
        int res = p1.getFamilyName().compareTo(p2.getFamilyName());
        if (res == 0) {
            res = p1.getFirstName().compareTo(p2.getFirstName());
        }
        return res;
    }
}
```

Uppgift 16

```
public int binarySearch( Object[] array, Object value, Comparator comp) {
    int res = -1, compare, min = 0, max = array.length - 1, pos;
    while( ( min <= max ) && ( res == -1 ) ) {
        pos = (min + max) / 2;
        compare = comp.compare( value, array[ pos ] );
        if( compare == 0 )
            res = pos;
        else if( compare < 0 )
            max = pos - 1;
        else
            min = pos + 1;
    }
    return res;
}
```

Uppgift 17

```
public Object[] getArray( int n, ObjectCreator creator ) {
    Object[] res = new Object[n];
    for(int i=0; i<res.length; i++) {
        res[i] = creator.nextObject();
    }
    return res;
}
```

Uppgift 18

```
public class Permutation implements ObjectCreator {
    private Laboration3 lab3 = new Laboration3();
    private ArrayList<String> word;
    int index = -1;

    public Permutation( String str ) {
        word = lab3.permute(str);
    }

    public Object nextObject() {
        index = ( index + 1 ) % word.size();
        return word.get( index );
    }
}
```