Föreläsning 2

Sökning

Linjär sökning

Likhet: ==, equals

Binär sökning

Comparable, Comparator

Skansholm: 118, 305-309, 380-397, 652-654

Linear search i osorterad lista

Problem: Sök efter ett ett bestämt värde i en array

Algoritm:

Kontrollera elementen i listan med start på element i postion 0.
    Om värdet på aktuellt element är det sökta värdet så
        returnera elementets position
Returnera -1

Halvkod:

Antag att elementet inte finns – sätt variabeln res till -1
För varje element i listan och så länge res == -1
    om element i position i har det sökta värdet så
        tilldela res elementets position (värdet på i)
Returnera res

eller

För varje element i listan
    om element i position i har det sökta värdet så
        returnera elementets position
Returenera -1

LinearSearch

==, jämföra enkla variablertyper

Implementering

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

eller

```
public int indexOf2( int[] array, int value ) {
        for( int i=0; i<array.length; i++ ) {
        if( value == array[ i ] ) {
            return i;
        }
    }
    return -1;
}
```

== används när man vill jämföra enkla variabletyper som int, long, double, boolean och char

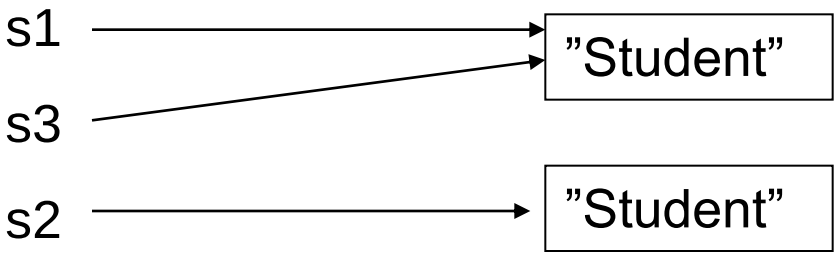
==, jämföra referensvariabler

== kan användas vid jämförelse av referensvariabler. Vad man då kontrollerar är om referensvariablerna refererar till samma objekt.

```
String s1 = new String( "Student" ), s2 = new String( "Student" ), s3;
s3 = s1;
System.out.println( "s1==s2: " + ( s1 == s2 ) );
System.out.println( "s1==s3: " + ( s1 == s3 ) );
```

Ger utskrifterna

```
s1==s2: false
s1==s3: true
```



Equals

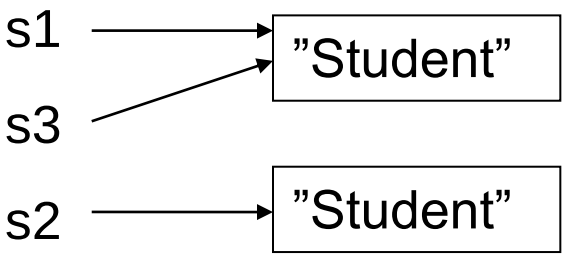
equals, jämföra referensvariabler

- **equals** kan användas vid jämförelse av referensvariabler. I detta fall anropas metoden equals med ett objekt och det andra är argument vid anropet.
- Vad som kontrolleras beror på equals-metoden i klassen. Om klassen inte innehåller en egen version av equals används den ärvda versionen.
- Klassen **Object** implementerar equals-metoden. Den fungerar på samma sätt som ==.

```
String s1 = new String( "Student" ), s2 = new String( "Student" ), s3;
s3 = s1;
System.out.println( "s1.equals(s2): " + s1.equals( s2 ) );
System.out.println( "s1.equals(s3): " + s1.equals( s3 ) );
```

Ger utskrifterna

```
s1.equals(s2): true
s1.equals(s3): true
```



Klassen Object

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

Klassen String (ungefärlig kod)

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        return sameChars(anObject) ;
    }
    return false;
}
```

Metoden equals överskuggas i RealNbr

```
public class RealNbr {
   private double value;

   public RealNbr(double value) {
      this.value = value;
   }

   public boolean equals(Object obj) {
      boolean res = (this==obj);
      if( !res && (obj instanceof RealNbr) ) {
         RealNbr t = ( RealNbr ) obj;
         res = ( this.value == t.value );
      }
      return res;
   }
}
```

RealNbr

Binary search – söka i sorterad lista

Att söka i en sorterad lista är ganska enkelt.

Antag att listan är ordnad växande: 11, 14, 19, 20, 21, 22, 24, 26, 29, 30

Algoritm:

Så länge (det finns fler element att söka bland) och (elementet inte hittats)
    Om det mittersta elementet är det sökta så
        lagra positionen
    Annars om det mittersta elementet är större än det sökta så
        upprepa sökningen på den undre halvan av element
    Annars om det mittersta elementet är mindre än det sökta så
        upprepa sökningen på den övre halvan av element

Returnera den lagrade positionen (-1 om inget värde påträffats)

Exempel:

1. Sök värdet 25 i {11, 14, 19, 20, 21, 22, 24, 26, 29, 30}: 25 > 21
2. Sök värdet 25 i {…, 22, 24, 26, 29, 30}: 25 < 26
3. Sök värdet 25 i {…, 22, 24, …}: 25 > 22
4. Sök värdet 25 i {…, 24, …}: 25 > 24
5. Returnera -1

Binary search – söka i sorterad lista

Implementering med while-loop:

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

BinarySearch

Shuffle

Binary search – söka i lista med objekt

Principen när man söker efter objekt i en sorterad lista (Object[]) är samma som för enkla variabeltyper. Men för att kunna sortera listan och för att kunna söka i listan krävs något av följande:

- Objekten i listan är naturligt jämförbara (implementerar Comparable + jämförbara)
- En hjälpklass som implementerar Comparator och som används vid sortering / sökning.

Klassen Integer implementerar Comparable:

```
public class Integer implements Comparable<Integer> {
   :
   public int compareTo( Integer obj ) {

   }
   :
}
```

En ordnad array med Integer-objekt går att söka binärt.

```
int res = Arrays.binarySearch( integerArray, new Integer( 19 ) );
```

Klassen RealNbr implementerar inte Comparable. Anropet

```
int res = Arrays.binarySearch( realnbrArray, new RealNbr( 19 ) );
```

ger ett körfel - `java.lang.ClassCastException: f2.RealNbr ...`

Binary search – söka i lista med objekt

Alla klasser implementerar inte Comparable. Och även om en klass implementerar Comparable så ordnas kanske objekten enligt fel princip (felaktig sorteringsordning).

Om man önskar sortera en array med RealNbr avtagande så duger inte den naturliga sorteringsordningen (växande). Men det går bra att skicka med ytterligare ett argument till sort-metoden:

Arrays.sort(realNbrArray, new Decrease());

där det andra argumentet ska vara en klass som implementerar Comparator.

```
public interface Comparator<T> {
    public int compare( T obj1, T obj2 );
}
-------------------------------------------------------------------------------
public class Decrease implements Comparator<RealNbr> {

    public int compare( RealNbr nbr1, RealNbr nbr2 ) {
        if( nbr1.getValue() > nbr2.getValue() )     // return -nbr1.compareTo( nbr2 );
            return -1;
        else if( nbr1.getValue() == nbr2.getValue() )
            return 0;
        else
            return 1;
    }

}
```

Decrease