Beispielprogramm:

```
class ArrayBsp2
{ static int[] a;
  static int[] b;
  static int zahl = 15;

  static void feldAusgabe(int[] feld)
  { for (int i=0; i<feld.length; i++) Out.print(feld[i] + " ");
    Out.println();
  } // feldAusgabe

  static void feldAenderung(int[] feld)
  {  feld[2] = 21; }

  static void zahlAenderung(int wert)
  {  wert = 21; }

  public static void main(String[] arg)
  { // Out.println ("b[0] = " + b[0]); --> NullPointerException
    a = new int[3]; // a[0] = 0; a[1] = 0; a[2] = 0; automatisch
    Out.print ("Feld a: ");  feldAusgabe(a);
    b = a; // kein Fehler
    Out.print ("Feld b: ");  feldAusgabe(b);
    a[0] = 17;
    Out.print ("Feld a: ");  feldAusgabe(a);
    Out.print ("Feld b: ");  feldAusgabe(b);
    Out.println("Feldveraenderung in Methode");
    feldAenderung(a);
    Out.print ("Feld a: ");  feldAusgabe(a);
    Out.print ("Feld b: ");  feldAusgabe(b);
    b = null;
    // Out.println("b[0] = " + b[0]); --> NullPointerException
    Out.println("Zahl = " + zahl);
    zahlAenderung(zahl);
    Out.println("Zahlveraenderung in Methode");
    Out.println("Zahl = " + zahl);
  } // main
} // class ArrayBsp2
```

Ausgabe:

```
Feld a: 0 0 0
Feld b: 0 0 0
Feld a: 17 0 0
Feld b: 17 0 0
Feldveraenderung in Methode
Feld a: 17 0 21
Feld b: 17 0 21
Zahl = 15
Zahlveraenderung in Methode
Zahl = 15
```

Erklärung:

1. Durch

   ```
   int[] a;
   int[] b;
   ```

   werden die Arrayvariablen `a` und `b` deklariert. Das Array gibt es noch nicht. Es wird schon Speicherplatz reserviert, aber nicht für das Array, sondern nur für `a` bzw. `b` zur Aufnahme der Speicher**adresse** für das noch anzulegende Feld. Der Inhalt ist jeweils `null`, d.h. es ist noch keine Adresse enthalten. Dementsprechend bekommt man mit

   ```
   Out.println ("b[0] = " + b[0]);
   ```

   beim Programmlauf eine *NullPointerException*, d.h. man versucht mit Variablen, die eine Speicheradresse enthalten muss, zu arbeiten, obwohl sie keine Speicheradresse enthält.

2. Mit

   ```
   a = new int[3];
   ```

   wird Speicherplatz für das Feld angelegt, alle Elemente von `a` werden automatisch auf `0` gesetzt und `a` enthält nun die Speicheradresse des neu angelegten Feldes. Die Variable `a` ist zu einem **Zeiger** oder **Pointer** geworden.

   Einen ähnlichen Effekt erhält man, wenn man auf dem Desktop von WINDOWS zwei Verknüpfungen auf ein Programm erzeugt. Wenn man nun das Programm verändert (z.B. updatet), zeigen beide Verknüpfungen auf das gleiche veränderte Programm.

3. Die Zuweisung

   ```
   b = a;
   ```

   führt zu keinem Fehler. Die Arrayvariable `b` erhält dadurch die gleiche Speicheradresse wie die Variable `a`. Beide Variable **zeigen** also nun auf das gleiche Feld.

4. Mit

   ```
   a[0] = 17;
   ```

   wird das Array an der nullten Position mit einem Wert beschrieben. Da aber `b` auf das gleiche Feld (gleiche Speicheradresse) zeigt, wird dadurch auch der Inhalt des Arrays zu `b` verändert.

5. Mit

   ```
   feldAenderung(a);
   ```

   wird der Zeiger `a` als Parameter der Methode `feldAenderung(int[] feld)` übergeben. Dementsprechend zeigt nun die **lokale Arrayvariable** `feld` auf das gleiche Feld. Also wirkt sich eine Änderung eines Feldelementes auch auf das übergebene Feld aus.

6. Mit

   ```
   b = null;
   Out.println("b[0] = " + b[0]); --> NullPointerException
   ```

   wird der Zeiger `b` auf `null` gesetzt, d.h. `b` zeigt nirgendwo hin. Dann führt die folgende Anweisung zum angegebenen Fehler beim Programmlauf.

7. Mit

   ```
   zahlAenderung(zahl);
   ```

   wird der tatsächliche Wert (Speicher**inhalt**) von `zahl` als Parameter der Methode `zahlAenderung(int wert)` übergeben bzw. dorthin kopiert. Dementsprechend enthält nun die **lokale Variable** `wert` den gleichen Inhalt wie die **globale Variable** `zahl`. Eine Änderung der lokalen Variable wirkt sich nun nicht auf die globale Variable `zahl` aus.

8. Will man die Elemente eines Arrays einem anderen zuweisen, so muss man es mit einer Schleife elementeweise tun.

9. Hat ein Array keinen Zeiger mehr, der darauf weist, so hängt es „in der Luft“. Dies bemerkt aber das Java-System und führt dann eine *automatische Speicherbereinigung* oder *Garbage Collection* durch.

10. In Java sind nur die einfachen Variablentypen wie `int`, `double`, `boolean`, usw. keine Zeigertypen. Objekte (ein Array ist ein Objekt) sind immer Zeigertypen.

    Im Vergleich zu anderen Programmiersprachen wie z.B. Pascal, Modula und C kann man aber nicht mit Zeigern rechnen. Die Zeigerarithmetik ist verboten, was die Sicherheit der Programme deutlich erhöht.