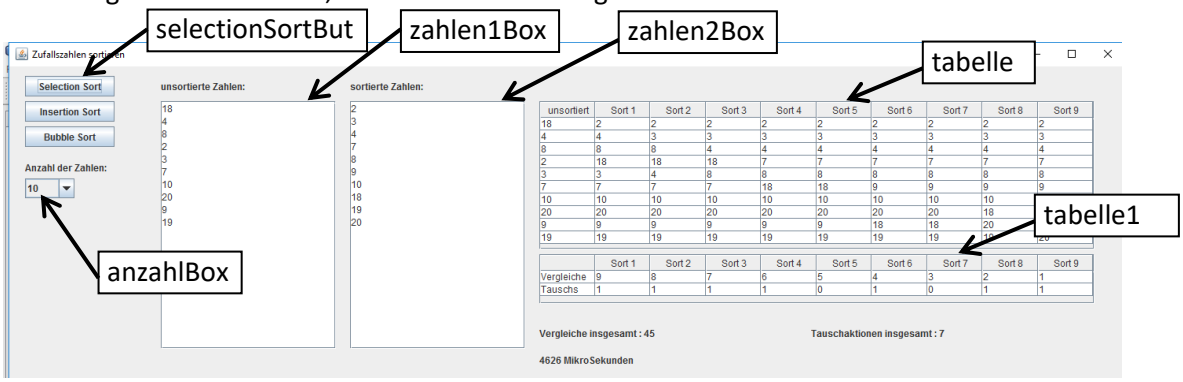
Projekt 10: Sortieren

Im letzten Projekt hast du in einer Datenmenge von 250 Namen gesucht. Wenn eine Firma mehr als eine Million Kunden hat und einen bestimmten Kunden in ihrer Datenbank sucht, kommt man so nicht weiter: die Suchzeiten wären viel zu lang und da in der Regel von den verschiedenen Mitarbeitern mehrere Suchanfragen gleichzeitig an eine Datenbank gestellt werden, würden die Wartezeiten viel zu lang werden. Da hilft nur eins: Die Daten müssen sortiert werden. In diesem Projekt lernst du verschiedene Sortieralgorithmen kennen, mit denen Zahlen möglichst schnell sortiert werden sollen.



Aufgabe 1: Kopiere das Projekt „ZahlenSortieren“ aus dem Schülerordner in deinen persönlichen Ordner. Das Projekt hat nur einie einzige Klasse namens **Gui** als Gui-Klasse. Die Oberfläche ist fertig implementiert wie oben dargestellt. Benötigt werden zwei JTextAreas **zahlen1Box** und **zahlen2Box** in denen die Zahlen zunächst unsortiert und schließlich sortiert erscheinen sollen. In der **anzahlBox** werden die AuswahlMöglichkeiten 10, 100, 1000, 10000 und 50000 für die Anzahl der zu sortierenden Zahlen angeboten. Den Code für die Auswahlmethode **anzahlBoxItemStateChanged** kannst du aus dem unteren Fenster kopieren: in der Variablen **anzahl** wird die Zahl der Zahlen gespeichert, die zufällig gezogen und später sortiert werden sollen. Als Feldvariablen wird ein Array definiert, das bis zu 100.000 Zufallszahlen speichern kann, einige weitere Variablen werden benötigt. Kopiere die Varibalendeklarationen direkt hinter die Klassendeklaration **public Class Gui extends javax.swing.JFrame{** .

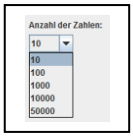
Methode **Vorbereiten**: erzeugt die Zufallszahlen

```
private void Vorbereiten(){
    zahlen1Box.setText("");
    zahlen2Box.setText("");
    boolean zahlGefunden,warSchonMal;
    int neueZahl;

    for (int i = 0; i < Anzahl; i++){
      zahlGefunden = false; //zahlGefunden wird true, wenn eine neue Zahl gefunden wurde
      while (zahlGefunden == false){
        neueZahl = (int)((Math.random()) * Anzahl * 2 + 1); // eine Zufallszahl
        int j = 0;
        warSchonMal = false; //warSchonMal wird true, wenn nochmals die gleiche Zahl gezogen wird
        while ((warSchonMal == false) && (j < i)){
          if (neueZahl == Zahl [j]) {warSchonMal = true;}
          else { j++;}
        }
        if (warSchonMal == false) {
          zahlGefunden = true;
          Zahl [i] =  neueZahl;
        }
      }
      zahlen1Box.append(String.valueOf(Zahl [i]) + "\n" );
    }
  }
```

Methode **anzahlBoxItemStateChanged** :

```
int Index;
      int [] ZahlGrenzen = new int []
{10,100,1000,10000,50000};
      Index = anzahlBox.getSelectedIndex();
      anzahl = ZahlGrenzen [Index];
```



Variablendeklarationen der Klasse Gui:

```
int[] Zahl = new int [100000];
int anzahl, gesvergleiche, gestausche;
```

Text kann kopiert werden !

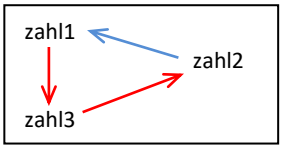
Durch Klick auf den Button selectionSortBut soll zunächst nur die Methode **Vorbereiten** aufgerufen werden, in der die Zufallszahlen gezogen werden. Wenn du das Programm jetzt startest, sollten in der **zahlenBox** 10 Zufallszahlen erscheinen, über die **anzahlBox** solltest du die Zahl der Zufallszahlen vergrößern können.

Aufgabe 2: Lies im Buch S. 151 f. den Artikel Selection Sort. Besorge dir das Arbeitsblatt, auf dem 10 vorgegebene Zahlen nach der Methode **SelectionSort** sortiert werden sollen. Fülle das Arbeitsblatt aus.

Aufgabe 3: Rechts ist im **Pseudocode** der Algorithmus **SelectionSort** dargestellt. In ähnlicher Form findest du ihn auch im Buch auf S. 153 oben, hier werden die Zufallszahlen jedoch ab Stelle 0 gespeichert und nicht wie bei uns ab Stelle 1. Implementiere die Methode **SelectionSort**, die die gezogenen Zufallszahlen sortiert. Die sortierten Zahlen sollen anschließend in der **zahlen2Box** sortiert ausgegeben werden.

Tipp: Wenn man zwei Zahlen **zahl1** und **zahl2** vertauscht, benötigt man immer eine dritte Variable z. B. **zahl3**. Der Tausch geht dann in drei Schritten:

zahl3 = zahl1;
zahl1 = zahl2;
zahl2 = zahl3;




```
SelectionSort (A : Array sortierbarer Elemente)
anzahl = Anzahl von A
für jedes links von 1 bis Anzahl - 1 wiederhole
  min = links
  für jedes j von links + 1 bis Anzahl widerhole
    wenn A[j] < A [min] dann
      min = j
    ende wenn
  ende wiederhole
  vertausche A[min] und A [links]
ende wiederhole
```

Nach dieser Zeile steht die nächste Zahl an der richtigen Stelle

Aufgabe 4: Um den Sortieralgorithmus besser zu verstehen, sollen die Zahlen jeweils nach einem Sortiervorgang in einer Spalte der Tabelle **tabelle** ausgegeben werden (s. letzte Seite Oben). Die Tabelle soll 10 Spalten und 10 Zeilen haben, hinzukommt eine Zeile mit Überschriften. Erzeuge die Tabelle **tabelle** wie in der Abbildung angegeben. Kopiere dazu den rechts dargestellten Code über die Methode **Customize Code…** und überschreibe die Standardeinstellungen von Netbeans mit je 4 Spalten und Zeilen. Dazu muss der zweite Eigenschaftseintrag von **default code** auf **custom property** umgestellt werden.
Unterhalb des Kopiercodes ist angegeben, wie die Einstellung der Tabelle aussehen sollte: Nach dem Programmstart sollte jetzt eine leere Tabelle mit den angegeben Überschriften dargestellt werden.

Ersetze in der Tabelle **tabelle** den Code über die Methode Customize Code:

```
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null},
{null, null, null, null, null, null, null, null, null, null}
},
new String [] {
   "unsortiert", "Sort 1", "Sort 2", "Sort 3","Sort 4", "Sort 5", "Sort 6", "Sort 7", "Sort
8", "Sort 9"
```

Eintrag muss verändert werden

So wird eine Tabelle mit 10 leeren Spalten und Zeilen definiert.

Und so setzt man innerhalb des Programmcodes einen Eintrag in einer Tabelle:
tabelle.setValueAt (String.valueOf(zahl [k]),k,m); schreibt in die k.te Zeile und die m. Spalte die Zufallszahl **zahl [k]**.

Was wird geschrieben? | Zeile | Spalte

Ergänze die Methode **SelectionSort**, so dass die Tabelle wie dein Arbeitsblatt gefüllt wird: In der ersten Spalte (Spaltennummer 0) sollen die unsortierten Zufallszahlen stehen, in der 2. Spalte (Spaltennummer 1) stehen die Zufallszahlen, nachdem die kleine Zahl nach Oben verschoben wurde usw. In der letzten Spalte stehen schließlich die sortierten Zufallszahlen. Eine Spalte muss jeweils komplett geschrieben werden, nachdem ein Sortiervorgang abgeschlossen wurde, d. h. nachdem die nächste Zahl an die richtige Stelle geschrieben wurde.

Aufgabe 5: Im Buch wird auf S. 153 f. ein anderes Sortierverfahren beschrieben: **Insertion Sort**.
Hole dir wieder das passende Arbeitsblatt und vollziehe den Sortieralgorithmus mit den gegebenen Beispielzahlen nach, indem du die Zahlen nach jeder neu einsortierten Zahl jeweils in einer Spalte aufschreibst.
Implementiere dann die alternative Sortiermethode, so dass man sie über einen zweiten Button aufrufen kann. Links ist der angepasste Pseudecode dargestellt (vgl. Buch S. 155 Oben, dort liegt die erste Zahl bei A[0]).

```
InsertionSort (A : Array sortierbarer Elemente)
anzahl = Anzahl von A
für jedes i von 2 bis Anzahl  wiederhole
  merke = A [i]
  j = i
  solange j > 1 und A [j-1] > merke wiederhole
    A [j] = A [j-1]
    j = j - 1
  ende wiederhole
  A[j] = merke
ende wiederhole
```

Aufgabe 6: Bereite einene kleinen Vortrag vor, indem du eines der beiden Sortierverfahren möglichst anschaulich anhand deines Arbeitsblattes erklärst. **Schreibe dazu im Heft Notizen auf, für jeden Befehl sollte eine Notiz vorhanden sein.**

Zusatzaufgabe 1: Implementiere zusätzlich die Suchmethode Bubble Sort (Buch S. 15 f.), die man mit einem dritten Button starten können soll.

Zusatzaufgabe 2: In der Tabelle **tabelle2** (s. Abbildung zu beginn des Projektes) sollen jeweils die Zahl der Vergleiche und die Zahl der Vertauschungen von Zahlen angegeben werden. Implementiere die notwendigen Ergänzungen für die drei Sortieralgorithmen.
Der Text rechts kann wieder kopiert weden, um die Tabelle einzurichten.

```
{null, null, null, null,null, null, null, null,null, null},
      {null, null, null, null,null, null, null, null,null, null}

   },
   new String [] {
      "", "Sort 1", "Sort 2", "Sort 3","Sort 4", "Sort 5", "Sort 6", "Sort 7", "Sort 8", "Sort 9"
   }
```

Zusatzaufgabe 3: In einem Label soll zusätzlich angegeben werden, wie viel Zeit die Algorithmen zum Sortieren benötigt haben. Dies wird insbesondere bei großen Zahlenmengen interessant. Du benötigst folgende Befehle: **zeitStart = System.nanoTime();** liest die Systemzeit in Nanosekunden (1000 Nanosekunden = 1 Mikrosekunde) aus. **zeitEnde = System.nanoTime();** nach Abschluss des Sortierverfahrens liest die vergleichszeit zum Abschluss. Durch einfache Differenzbildung kann die benötigte Zeit in Nanosekunden berechnet werden. Gib die benötigte Zeit in Mikrosekunden an (1 Million Mikrosekunden = 1 Sekunde).