Sortieralgorithmus

JAVA-PROJEKT

Lars Bodewig
BS AWIS 17 | LF 6

Inhalt

Einführung

Im Rahmen der Leistungserbringung im LF 6 – Entwickeln und Bereitstellen von Anwendungssystemen soll ein selbstentworfenes und –geschriebenes Java-Projekt zur Umsetzung eines Sortieralgorithmus erarbeitet werden. Die Kriterien umfassen die Sortierung von alphanumerischen Zeichen, den ‚Single-Step-Betrieb', die Eingabe von zu sortierenden Zeichen durch den Anwender und eine grafische Abbildung des Sortierverfahrens in einer Oberfläche.

Als persönliches Ziel wurde zusätzlich die Modularität der Komponenten und ein hoher Abstraktionsgrad der Anwendung definiert und wie folgend beschrieben umgesetzt.

Für die verwendete Sprache Java in der Version 8 wird die Entwicklungsumgebung Eclipse Photon verwendet.

Core-Klassen

Um eine hohe Modularität zu erreichen, die erlaubt die Komponenten universell einzusetzen, ist ein hoher Abstraktionsgrad nötig. In Java lässt sich diese Abstraktion durch Verwendung von abstrakten Klassen, Interfaces und generischen Inhaltstypen umsetzen.

Während abstrakte Klassen und Interfaces Implementierungsrichtlinien für Klassen vorgeben und somit Schnittstellen für die Verwendung dieser darstellen setzt die Verwendung von sogenannten ‚Generics' auf die Anwendbarkeit bestimmter Funktionen oder Klassen auf beliebige Inhaltstypen in einem definierten Rahmen. Weit verbreitete Anwendung dieser sind in der nativen Java- Collections-Bibliothek zu finden, in der beispielsweise Listen beliebige Inhaltstypen verwenden und somit eigene Klassen ebenfalls in dieser Datenstruktur abgebildet und verarbeitet werden können.

Das Package Core bietet diese Abstrahierung durch das Interface Numerical für Inhaltstypen zu sortierender Elemente, die abstrakte Klasse Algorithm zur Abdeckung bei Algorithmen gemeiner Funktionen und Definition einer Schnittstelle für Sortieralgorithmen und die Klasse Dialog, die für beliebige Inhaltstypen eine Eingabe über einen Dialog abfragt und verarbeitet zu sortierenden Elementen.

Numerical

```
package core;

public interface Numerical<T, E> extends Comparable<T> {

	// verarbeitet den Eingabe-String aus dem Dialog und erzeugt mit dem Pattern
	// ein Array von Inhaltsobjekten
	public T[] processInput(String input);

	// gibt das Pattern für die Dialog-Eingabe zurück
	public String getPattern();

	// gibt den numerischen Wert des Inhaltsobjekts zurück
	public int numericalValue();

	// gibt das Inhaltsobjekt als String zurück
	public String toString();

	// gibt eine Deep Copy eines Arrays von Inhaltsobjekten zurück
```

```
	public T[] clone(T[] array);
}
```

Der doppelt generische Inhaltstyp erlaubt die Übergabe der implementierenden Klasse und des Inhaltsobjekts der implementierenden Klasse:

```
public class Char implements Numerical<Char, Character> {
```

Dies ist nötig um die im Interface Numerical vererbte compareTo-Methode auf der implementierenden Klasse verwenden zu können und gleichzeitig generische Inhaltsobjekte in der implementierenden Klasse zu erlauben.

Die Methode numericalValue() gibt dem Interface seine namensgebende Eigenschaft: ein gewrapptes Inhaltsobjekt muss seinen numerischen Wert zurückgeben können, um in einem Balkendiagramm darstellbar zu sein.

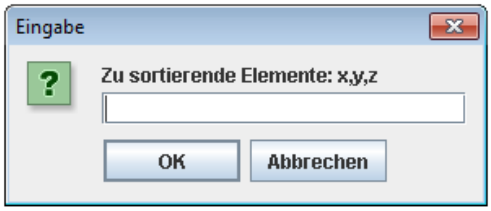
Die clone-Methode erlaubt eine Deep Copy des Inhaltsobjekts zu erstellen, da die Übergabe von Objekt-Referenzen statt Objekt-Values in Java eine Kopie des Objekts voraussetzt, um in der Historie unverändert zu bleiben.

Zur Eingabe von Inhaltsobjekten über ein Dialogfeld muss eine objektspezifische Verarbeitung ermöglicht werden, da beispielsweise numerische Werte im Gegensatz zu alphanumerischen Zeichen mehrstellig sein können. Die Methode getPattern() wird daher die im Dialog angezeigte Semantik für die Eingabe darstellen und processInput(String) die Implementierung der Verarbeitung des Eingabe-Strings in einen Array von Inhaltsobjekten voraussetzen.

Dialog

```
package core;

import javax.swing.JOptionPane;

public class Dialog {

	// zeigt einen Dialog mit dem Pattern des Sortieralgorithmus an und gibt
	// einen Array von vom Inhaltstyp verarbeiteten Objekten zurück
	public static <T extends Numerical<T, ?>> T[] getElements(T numerical) {
		String elements = JOptionPane.showInputDialog("Zu sortierende
		Elemente: " + numerical.getPattern());
		return numerical.processInput(elements);
	}
}
```

Durch die generische Methode getElements(T) der Klasse Dialog ist es möglich beliebige Inhaltstypen als Input vom User anzunehmen und zu verarbeiten. Dies geschieht durch den Aufruf der Methoden der von Numerical erbenden Klassen getPattern() und processInput(String), die ein Format zur Eingabe-Syntax und Ausgabe des Eingabe-Strings in einen T[] erlauben. Durch die Verarbeitung in der Inhaltstyp-Klasse selber wird bewusst die Java-Limitation bei der Erstellung von generischen Arrays umgangen, da der Typ in der implementierenden Klasse bekannt ist.



Ein Inhaltstyp, der alphanumerische Zeichen abbildet, könnte beispielsweise ein Pattern „x,y,z“ vorgeben und die Eingabe durch die Trennung des Strings an den Kommas verarbeiten.

Die Implementierung eines solchen Inhaltstyps ist im Kapitel Char weiter beschrieben.

Algorithm

Die abstrakte Klasse Algorithm des package core bietet abstrakte und bereits implementierte Methoden zur schnellen, schlanken Implementierung eines beliebigen Algorithmus, da wichtige allgemeine Funktionen zur Überprüfung der Sortierung, zur Abbildung einer Historie, zur Ausgabe der Elemente als String und Initialisierung der Klasse im Konstruktor bereits vorgegeben werden.

Lediglich der Namen und die Beschreibung des Algorithmus müssen and den Super-Konstruktor übergeben und die Ausführung eines einzelnen Sortierschritts von einer erbenden Klasse implementiert werden, um einen voll funktionsfähigen Algorithmus mit Historie zur Erzeugen.

```
package core;

import java.util.Stack;

public abstract class Algorithm<T extends Numerical<T, ?>> {

	// der Name des Algorithmus
	private String name;

	// die zu sortierenden Elemente vom Typ T
	protected T[] elements;

	// Verlauf der Zustände des Arrays von zu sortierenden Elementen
	private Stack<T[]> history;

	// die Sortiertreihenfolge
	private boolean ascending;

	// erzeugt ein neues Algorithmus-Objekt
	public Algorithm(String name, T[] elements, boolean ascending) {
		this.name = name;
		this.elements = elements;
		this.ascending = ascending;
		history = new Stack<T[]>();
	}

	// gibt den Namen des Algorithmus zurück
	public String getName() {
		return name;
	}

	// gibt die zu sortierenden Elemente zurück
	public T[] getElements() {
		return elements;
	}

	// gibt zurück, ob ein Schritt rückwärts gemacht werden kann
	public boolean canStepBackwards() {
		return history != null && !history.isEmpty();
	}

	// geht einen Schritt in der Sortierung zurück
```

```
	public void stepBackwards() {
		if (canStepBackwards()) {
			elements = history.pop();
		}
	}

	// gibt zurück, ob elements sortiert ist
	public boolean isSorted() {
		boolean isSorted = true;
		for (int i = 0; i < elements.length - 1; i++) {
			if (ascending && elements[i].compareTo(elements[i + 1]) > 0
					|| !ascending && elements[i].compareTo(elements[i
					+ 1]) < 0) {
				isSorted = false;
			}
		}
		return isSorted;
	}

	// führt einen einzelnen Sortierschritt aus
	public void stepForward() {
		if (!isSorted()) {
			history.push(elements[0].clone(elements));
			step();
		}
	}

	// gibt einen String mit allen zu sortierenden Elementen zurück
	@Override
	public String toString() {
		StringBuilder sb = new StringBuilder(elements.length);
		for (T t : elements) {
			sb.append(t.toString());
		}
		return sb.toString();
	}

	// führt einen einzelnen Sortierschritt aus
	protected abstract void step();

	// gibt die Beschreibung des Algorithmus zurück
	public abstract String getDescription();
}
```

In einem generischen Stack werden die verschiedenen Zustände der Inhaltselemente als Historie abgebildet, in der Schritte zurück gemacht werden können. Die Methoden canStepBackwards() und stepBackwards() automatisieren diese Funktionalität vollständig, ohne Ergänzungen in der erbenden Klasse.

Die Ausgabe aller Elemente als String erfolgt durch die Konkatenierung der vom Element zurückgegebenen Strings der in Numerical vorgegebenen toString()-Methode. Ob die Elemente fertig sortiert wurden, kann mit der Methode isSorted() überprüft werden, die die im Konstruktor festgelegte Sortierreihenfolge berücksichtigt.

Ein Implementierungsbeispiel für einen Algorithmus ist im Kapitel BogoSort zu finden.

Inhaltstypen und –objekte

Die Inhaltstypen, Implementationen der abstrakten Klasse Algorithm und des Interface Numerical, sind im package algorithm abgebildet. Beispielsweise wurden BogoSort und SelectionSort und ein Inhaltstyp für characters implementiert.

Char

```
package algorithm;

import core.Numerical;

public class Char implements Numerical<Char, Character> {

	// das Inhaltsobjekt vom Typ Character
	private final Character content;

	// das im Dialog angezeigte und in processInput(String) verarbeitete Pattern
	public static final String pattern = "x,y,z";

	// Erzeugt einen neuen Char aus einem primitiven Character
	public Char(char content) {
		this.content = new Character(content);
	}

	// gibt den numerischen Wert zurück
	@Override
	public int numericalValue() {
		return (int) content;
	}

	// gibt zurück, ob ein anderer Char größer, kleiner oder gleich diesem ist
	@Override
	public int compareTo(Char arg0) {
		return content.compareTo(arg0.content);
	}

	// verarbeitet den Eingabe-String aus dem Dialog und erzeugt mit dem Pattern
	// ein Char-Array
	@Override
	public Char[] processInput(String input) {
		String[] frag = input.split(",");
		Char[] cs = new Char[frag.length];
		for (int i = 0; i < frag.length; i++) {
			cs[i] = new Char(frag[i].trim().charAt(0));
		}
		return cs;
	}

	// gibt das Pattern für die Dialog-Eingabe zurück
	@Override
	public String getPattern() {
		return pattern;
	}

	// gibt das Inhaltsobjekt als String zurück
	@Override
	public String toString() {
		return content.toString();
```

```
	}

	// gibt eine Deep Copy eines Char-Arrays zurück
	@Override
	public Char[] clone(Char[] array) {
		Char[] clone = new Char[array.length];
		for (int i = 0; i < array.length; i++) {
			clone[i] = new Char(array[i].content);
		}
		return clone;
	}
}
```

Die Klasse Char ist eine konkrete Implementierung des Numerical-Interface zur Abbildung von alphanumerischen Zeichen. Deren numerischer Wert wird anhand des ASCII-Wertes festgelegt und mit der numericalValue-Methode zurückgegeben. Da Character-Objekte das Comparable-Interface bereits implementieren, wird dieses in der Char-Klasse über den Aufruf der Methode vom Inhaltsobjekt implementiert.

Zur Verwendung der Dialog-Klasse wird ein Pattern „x,y,z“ und von der Methode processInput(String) ein Array von Objekten der Klasse selber zurückgegeben. Die Eingabe wird dabei separiert durch Kommas und um Leerzeichen getrimmt, um Input des Patterns „x, y, z“ zu erlauben. Ebenso wird die Eingabe von mehreren Zeichen zwischen Kommas abgefangen durch die Beachtung von lediglich dem ersten Zeichen je zwischen zwei Kommas.

BogoSort

BogoSort ist ein nicht-linearer Sortieralgorithmus, dessen Laufzeit im Mittel O(n· n!) beträgt und damit ein Beispiel für ein sehr schlechtes Verfahren ist. Die Sortierung geschieht hierbei zufällig durch den Tausch zweier Elemente, wodurch die Laufzeit exponentiell mit der Anzahl der Elemente steigt.

```
	@Override
	protected void step() {
		int posAlt = (int) (Math.random() * elements.length);
		int posNeu = (int) (Math.random() * elements.length);
		T temp = elements[posAlt];
		elements[posAlt] = elements[posNeu];
		elements[posNeu] = temp;
	}
```

Dieses Verhalten wird in der Methode step() abgebildet, die zustandslos zwei Zufallspositionen im Element-Array wählt und die Elemente tauscht.

SelectionSort

Ein nicht zustandsfreier Algorithmus wie SelectionSort, der nach Angabe der Sortierreihenfolge auch MinSort oder MaxSort genannt wird, muss zu der step-Methode ein Attribut der aktuellen Position definieren.

```
	// die aktuelle Position im Array
	private int pos;

	// führt einen einzelnen Sortierschritt aus
	@Override
	protected void step() {
```

```
            int minPos = pos;
            for (int j = pos + 1; j < elements.length; j++) {
                  if (elements[j].compareTo(elements[minPos]) < 1) {
                        minPos = j;
                  }
            }
            T temp = elements[minPos];
            elements[minPos] = elements[pos];
            elements[pos] = temp;
            pos++;
      }
```

SelectionSort, ein naiver Algorithmus der Laufzeit $O(n^2)$, sucht von links nach rechts das kleinste (MinSort) oder größte (MaxSort) Element und tauscht es mit der aktuellen Position im Array, die von links beginnend bis zum rechten Rand verschoben wird. Um dies über die Einzelschritte transient darstellen zu können, ist der Zustand als Attribut gespeichert, nötig.

In beiden Fällen reicht diese Implementierung bereits aus einen vollständigen Algorithmus mit Prüfung der Sortierung, Ausgabe der Elemente, Abbildung einer Historie und Zurückspulen zu erstellen.

Das Säulendiagramm – API

Die BarChart-API ist in das Projekt eingebunden und mit der Wrapper-Klasse EasyBarChart implementiert. Eine Wrapper-Klasse kapselt eine Schnittstelle, um die konkrete Verwendung zu Vereinfachen oder anzupassen. Die Klasse EasyBarChart ermöglicht die schlanke Erzeugung eines Säulendiagramms aus einem Array von Numerical-Elementen und dem Namen des Algorithmus.

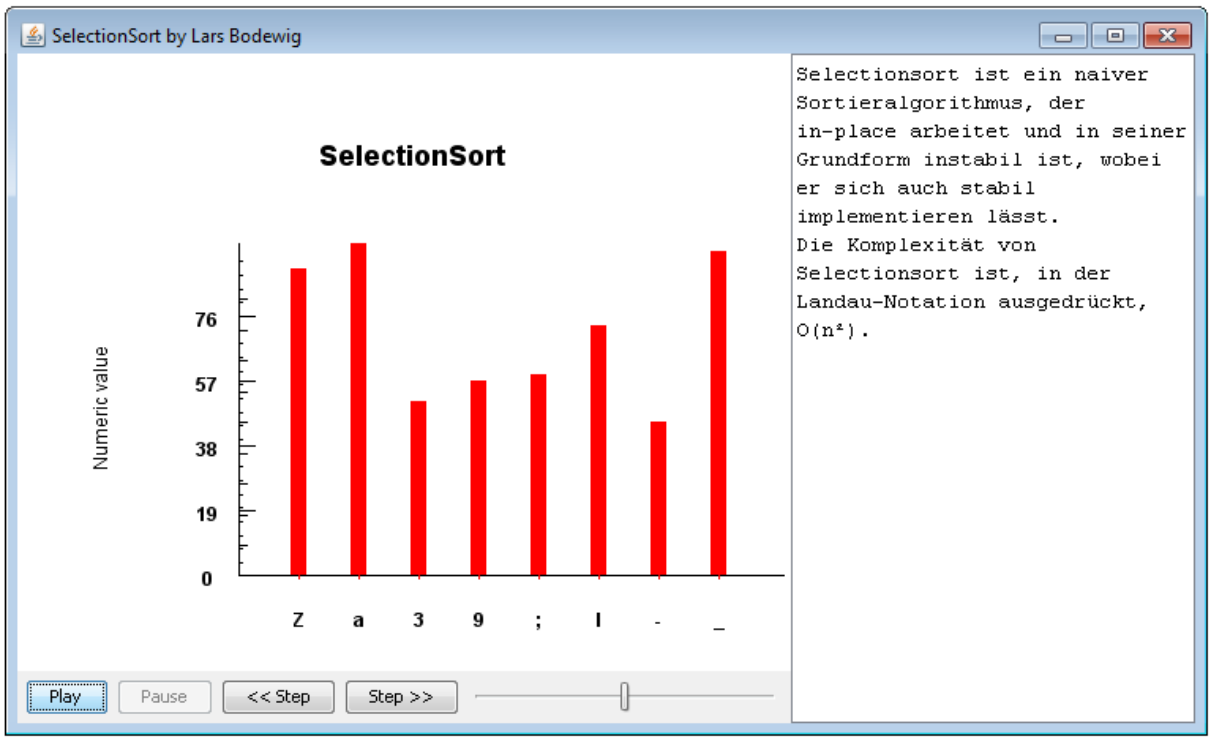
Über setBars(Numerical[]) wurde die Funktion der API erweitert und die Anpassung der Säulen eines bereits vorhandenen Diagramms ermöglicht, da zuvor nur das Erzeugen eines neuen Diagramms bei Änderung der Werte möglich war.

```
      // erzeugt ein Säulendiagramm aus dem Array von Inhaltsobjekten
      public static <T> EasyBarChart buildBarChart(Numerical<?, T>[] elements,
            String title) {
            int max = 0;
            ArrayList<Bar> values = new ArrayList<>();

            for (Numerical<?, T> t : elements) {
                  if (t.numericalValue() > max) {
                        max = t.numericalValue();
                  }
                  values.add(new Bar(t.numericalValue(), Color.RED,
                        t.toString()));
            }
            Axis yAxis = new Axis(max, 0, max / 5, max / 10, max / 20,
                  "Numeric value");

            return new EasyBarChart(values, yAxis, title);
      }

      // aktualisiert die abgebildeten Säulen anhand des Arrays von
      // Inhaltsobjekten
      public <T> void setBars(Numerical<?, T>[] elements) {
            int max = 0;
            ArrayList<Bar> values = new ArrayList<>();
```

```
            for (Numerical<?, T> t : elements) {
                  if (t.numericalValue() > max) {
                        max = t.numericalValue();
                  }
                  values.add(new Bar(t.numericalValue(), Color.RED,
                        t.toString()));
            }
            this.bars = values;
            this.yAxis = new Axis(max, 0, max / 5, max / 10, max / 20,
                  "Numerical value");
            this.repaint();
      }
```

Die Y-Achse wird dynamisch anhand des größten dargestellten Wertes skaliert, die Säulen-Farbe sowie die Y-Achsen-Beschriftung sind fest vorgegeben. Die Höhe der einzelnen Elemente wird durch den von numericalValue() zurückgegebenen Wert definiert und obliegt somit der Implementation der Numerical implementierenden Klasse. Nach einem Sortierschritt wird die setBars(Numerical[])-Methode aufgerufen, die ein repaint() der Frame-Komponente antriggert, sodass jeder Schritt grafisch dargestellt wird – Unabhängig der Standard-Aktualisierungsrate und der gewählten Sortiergeschwindigkeit.

Die Oberfläche



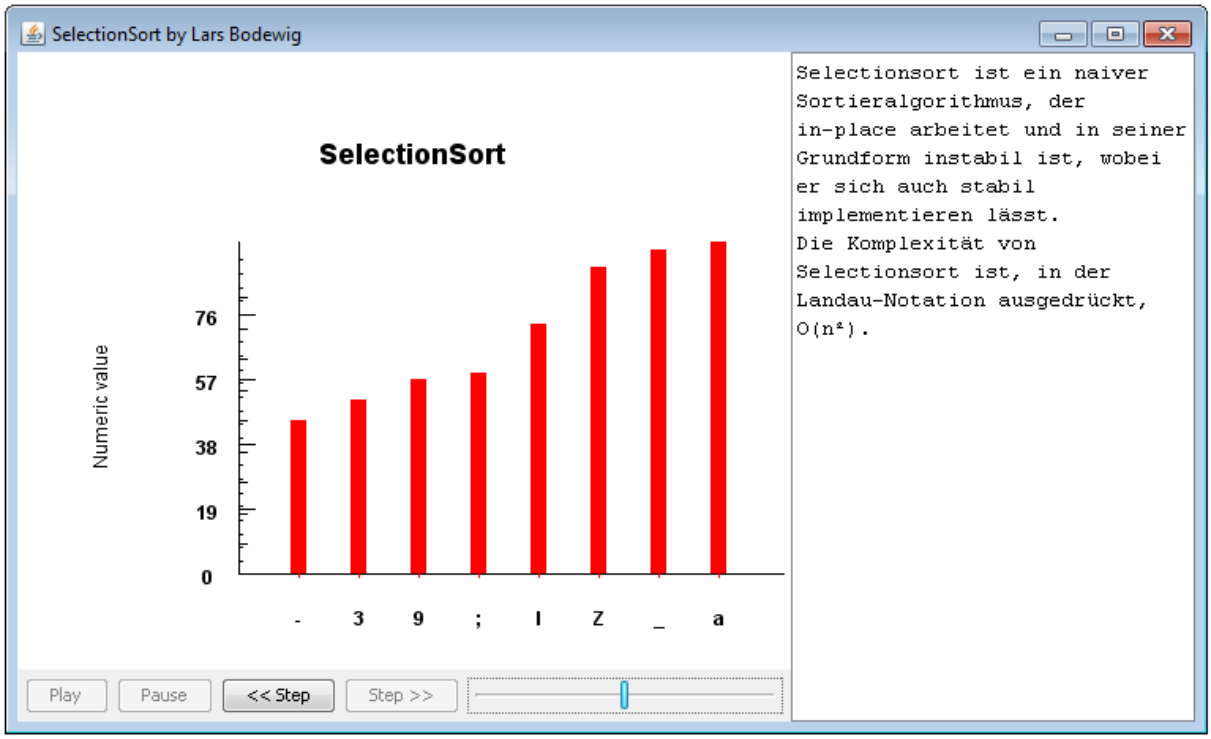
Das Frame teilt sich in drei feste Bestandteile: die Beschreibung, die Steuerung und die Visualisierung. Die Beschreibung bildet den zurückgegebenen String der getDescription()-Methode der Algorithmus-Klasse. Die Steuerung umfasst

- einen Play-Button zum Starten der automatischen Sortierung,
- einen Pause-Button zum Anhalten der automatischen Sortierung
- einen <<Step-Button, um einen einzelnen Sortierschritt zurück zu gehen,
- einen Step>>-Button, um einen einzelnen Sortierschritt vorwärts durchzuführen und

- einen Slider zur Anpassung der Geschwindigkeit der automatischen Sortierung.

Je nach Zustand der Anwendung (Ausführung der automatischen Sortierung, Ausgangszustand, sortierter End-Zustand) werden verschiedene Buttons de- oder aktiviert und können somit nur genutzt werden, wenn die Logik es erlaubt.

Das Säulendiagramm stellt die Elemente als Säulen mit der Höhe der numericalValue()-Methode dar, als Beschriftung des Elements wird die toString()-Methode abgefragt. Der Titel des Diagramms und des Fensters bilden den Namen des Algorithmus ab.



Event Handling

Zur Steuerung der Sortierung über die grafische Oberfläche sind ActionListener nötig, die bei Button-Events ausgelöst werden; diese Methoden sind in der Klasse EventHandler angelegt.

Um die Steuerung und Aktualisierung der Oberfläche zu bewahren bei Ausführung langwieriger Tasks wie dem automatischen Sortieren, müssen die Events asynchron behandelt werden. In Java wird dies über Threads abgebildet, denen man Runnable-Objekts, auszuführender Code, übergibt und startet.

Da im weiteren Sinne jede Interaktion mit der UI über die Steuerung lange dauern kann – dies ist der Implementierung des genutzten Sortieralgorithmus überlassen – werden alle Button-Events über eine Faktorisierungsmethode asynchron in einem Hintergrundthread gestartet.

```
// ruft die übergebene Methode asynchron in einem neuen Thread auf
public static ActionListener createListener(Runnable r) {
	return (e) -> new Thread(r).start();
}
```

Über eine neue Notation seit Java 8 können Methoden als Parameter übergeben werden, was einem Runnable-Objekt entspricht. Die Faktorisierungsmethode kann demnach so im der UI verwendet werden, um einen ActionListener auf einen Button zu registrieren, der bei einem Klick ausgeführt wird.

```
playButton.addActionListener(EventHandler.createListener(eventHandler::play));
```

Die Methode setButtonState(int, boolean) in der Application-Klasse ermöglicht das Steuern des Button-Zustandes in der UI ohne diese als Attribute zu public ändern zu müssen, was jeder anderen Klasse die Manipulation der Darstellung und Aktion des Buttons ermöglichen würde.

```
// aktiviert oder deaktiviert einen Button
public void setButtonState(int button, boolean enabled) {
	switch (button) {
	case 0:
		playButton.setEnabled(enabled);
		break;
	case 1:
		pauseButton.setEnabled(enabled);
		break;
	case 2:
		stepBackButton.setEnabled(enabled);
		break;
	case 3:
		stepForwardButton.setEnabled(enabled);
		break;
	}
}
```

Bei Drücken des Play-Buttons wird folgende Methode als ActionListener faktorisiert und asynchron ausgeführt:

```
// startet die automatische Ausführung des Sortieralgorithmus
public void play() {
	running = true;
	app.setButtonState(0, false);
	app.setButtonState(1, true);
	app.setButtonState(2, false);
	app.setButtonState(3, false);
	while (!algorithm.isSorted() && running) {
		algorithm.stepForward();
		app.updateBarChart();
		try {
			Thread.sleep(app.getSpeed());
		} catch (InterruptedException e) {
			System.out.println(e);
		}
	}
	running = false;
	app.setButtonState(1, false);
	if (algorithm.isSorted()) {
		app.setButtonState(0, false);
		app.setButtonState(3, false);
	} else {
		app.setButtonState(3, true);
	}
	if (algorithm.canStepBackwards()) {
		app.setButtonState(2, true);
	}
}
```

Neben der Änderung der Button-States durch den geänderten Zustand zum automatischen Sortieren wird eine Schleife gestartet, die in getSpeed()-Intervallen einen Sortierschritt ausführt und das Säulendiagramm aktualisiert.

Fazit

Abschließend ist zu sagen, dass der zeitliche Rahmen sehr knapp gewählt war, bis zum Abgabetermin wurden die Unterrichtszeit sowie einige Pausen stringent ausschließlich für das Projekt verwendet, was Ausfälle sehr ärgerlich machte.

Das selbstgesetzte Ziel der Modularität und Abstrahierung zur einfachen, dynamischen Implementierung von weiteren Sortieralgorithmen wurde erreicht durch den Mehraufwand der generischen Inhaltstypen und abstrakten Klassen.

Die gewählte API ist hinreichend und musste durch eine eigene Wrapper-Klasse um die benötigte Funktionalität erweitert werden, die Visualisierung hingegen klappt sehr gut und ist sauber strukturiert. Der Single-Step-Betrieb wurde vorwärts und rückwärts implementiert über die gesamte Zustandshistorie des Sortierverfahrens. Die grafische Oberfläche enthält eine einfach anpassbare Beschreibung des Algorithmus und den Namen des Autors, bei Ausführung der automatischen Sortierung bleibt die UI reaktiv durch die Verwendung von asynchronen Hintergrundthreads.

Die Dokumentation beschreibt die Funktionalität gegliedert nach packages und umfasst die wichtigsten Teile des Codes, der vollständig kommentiert ist; es werden verschiedene Sortieralgorithmen beschrieben und implementiert, die testweise im beigelieferten Projekt ausgeführt werden können. Eine Zeitplanung als Gantt-Diagramm wurde separat geführt und als Excel-Tabelle mitgeliefert.