

Java Stichwortverzeichnis

22. Januar 2020

Kategorie	Unterkategorie	Beschreibung	Skript
Arrays	Initialisierung	• <code>Datentyp[] myArray = new Datentyp[length];</code>	1D/4
	Indizes	• Zugriff auf Speicherort, startet bei [0]	
	Literal null	• Referenz auf keine Speicheradresse, führend ins 'Nichts'	1D/17
	Längenabfrage	• <code>myArray.length;</code>	1D/24
	Durchlauf	• <code>for(int i = 0; i < myArray.length; i++) {};</code> • Kurzform: <code>for(Datentyp myNewArray : myArray) {}</code> • statt <code>myArray[i]...; → myNewArray...;</code>	1D/26
	Gemischte Klassen	• Mithilfe von Vererbung möglich (statischer/dynamischer Typ)	1F/89

Arithmetrische Ergebnistypen	Kombinationen	<ul style="list-style-type: none"> • Kombination immer int, außer einer der Operanden ist long • falls float vorhanden, dann Ergebnis float (selbiges für double) • <code>double > float > long > int</code> 	1B/205
------------------------------	---------------	--	--------

Arithmetrische Operationen	Regeln	• Punkt vor Strich	1A/106
	Division bei int	<ul style="list-style-type: none"> • $2 + 7/3 = 4$, falls alle Zahlen vom Datentyp int sind • \rightarrow Abschneiden der Nachkommastelle 	1A/107
	modulo	• Ganzzahlige Division mit Rest (Beispiel: $11 \bmod 4 = 3$)	1A/108
	Kurzformen	<ul style="list-style-type: none"> • <code>i = i + 5; → i += 5; (-=, *=, /=, %=)</code> • <code>i = i + 1; → i++;</code> 	1B/194
	binär und Infix	<ul style="list-style-type: none"> • Binär: jeder zwei Operanden • Infix: Operator zwischen Operanden • Operationen: <code>+</code> <code>-</code> <code>*</code> <code>/</code> <code>%</code> 	1B/195
	unär und präfix	<ul style="list-style-type: none"> • Unär: nur ein Operand • Präfix: Operator vorangestellt • Vorzeichen: <code>+</code> <code>-</code> 	1B/196
	Inkrement Dekrement	<ul style="list-style-type: none"> • Unär und Postfix (nachgestellt) <code>i++</code>; <code>i--</code>; • Auch als Präfix: <code>++n</code>; <code>--n</code>; → Zurückliefern des neuen Werts von n <ul style="list-style-type: none"> • erst n, dann erhöhen erst erhöhen, dann n 	1B/197
	Zuweisungs-basiert	• <code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code>	1B/198
	Vergleiche	• <code>==</code> <code>!=</code> <code><</code> <code>></code> <code><=</code> <code>>=</code>	1B/199
	Ternär	<ul style="list-style-type: none"> • Bedingungsoperator: <code>'?'</code> • z.B.: <code>x = (Bedingung b) ? Falls b true : Falls b false</code> 	1B/226

Begriffs-erklärung	Variablen	• Container, deren Werte veränderbar sind (z.B.: Integer, myVariable,...)	1B/95
--------------------	-----------	---	-------

	Referenz	<ul style="list-style-type: none"> Dadurch wird ein Objekt referenziert (Verweis) (z.B.: Name eines Objekts von Klassen, Arrays,...) 	1B/96
	Objekt	<ul style="list-style-type: none"> Der Teil, der mit dem Operator new reserviert wird Objekte != Variablen/ Konstanten und andersrum 	
	Konstante	<ul style="list-style-type: none"> Syntax: final ...; Wert kann nicht mehr geändert werden Referenzen auch nicht änderbar (referenziertes Objekt schon) 	1B/100

Bytedaten	Eigenschaften	<ul style="list-style-type: none"> bytweweises Lesen bzw Schreiben in Dateien in Praxis jedoch wahrscheinlich irrelevant → Binärdateien einfacher über Bibliotheken zugänglich hier sinnvoll bei Zugriff auf Bilder oder Videos 	8/134
	InputStream	<ul style="list-style-type: none"> Abstrakte Klasse z.B. FileInputStream für Lesezugriff auf Dateien Subtypen: <ul style="list-style-type: none"> → java.util.zip.ZipInputStream (für Zip-Dateien) → java.util.jar.JarInputStream (für Jar-Dateien) → javax.sound.sampled.AudioInputStream 	8/142
	Einlesen	<ul style="list-style-type: none"> Einlesen mithilfe While-Schleife (z.B.: while(true){} Speichern als int: int n = InputStream.read(); Erreichen des Dateiendes: if (n == -1) { return; } -1 als Konvention für unmöglichen Wert 	8/135
	Schreiben	<ul style="list-style-type: none"> Ziel eines OutputStream: Datensenke Analog zu InputStream Methode FileOutputStream.write(int); 	8/145
	Buffered-Inputreader	<ul style="list-style-type: none"> Kann mehrere Bytes auf einmal einlesen Erhält intern ein Array von Bytes und gibt immer eins davon aus (Puffer) Wenn Array leer, wird neues Array eingelesen Bei Write genau umgekehrt, Byte wird in Puffer zwischengespeichert → Sobald Puffer voll → komplett geschrieben 	8/150
	PrintStream	<ul style="list-style-type: none"> Dient als Konvertierer in byteweise Repräsentation System.out: Klassenattribut out ist von Klasse PrintStream System.in: Klassenattribut in ist von Klasse InputStream System.err: Separates Log für Fehlerdateien → Um z.B. Fehlermeldungen aus anderem Output zu filtern (Log-Datei) 	8/157
		<ul style="list-style-type: none"> 	

Collections	Definition	<ul style="list-style-type: none"> Sammlungen von Elementen eines bestimmten Typs Typ als generischer Typparameter offen alle im Package java.util 	7/2
	Zentrale Elemente	<ul style="list-style-type: none"> Collection (Interface): → Alle Collections implementieren dieses Interface Collections (Klasse): → nützliche Basisalgorithmen (Sortieren) List (Interface) → Erweitert Collection (z.B.: Reihenfolge auf Elementen) → Elemente in Reihenfolge (definierte Position) Iterator (Interface): → Durchlauf aller Elemente einer Collection Beispiele Collection-Klassen: → Vector, LinkedList, ArrayList, TreeSet, HashSet 	7/3
	Syntax	<ul style="list-style-type: none"> Collection<Number> c1 = new ArrayList<Number>; → Variable vom Interface Collection instanziiert mit Number → Objekt von Klasse ArrayList (implementiert Collection) 	7/10

Collection	<ul style="list-style-type: none"> • boolean add (T element) throws ... → Hinzufügen von Elementen zur Liste / true falls erfolgreich • addAll, Parameter Typ Collection → Fügt alle Objekte aus übergebener Collection in Aufruf ein • size → Gibt Anzahl der Elemente in Collection aus • isEmpty → Liefert dann true, wenn Collection keine Elemente enthält • contains, Parameter Typ Object → überprüft ob Object in Collection enthalten ist • containsAll, Parameter Typ Collection → true, falls contains für alle Elemente der Collection true • clear → Entfernt alle Elemente aus der aufrufenden Collection • remove, Parameter Typ Object → Liefert true, falls Object vorhanden und entfernt dieses 	7/18
List	<ul style="list-style-type: none"> • indexOf, Parameter Typ Object → Liefert ersten Index zurück, an dem Object zu finden ist → Liefert -1 zurück, falls Object nicht enthalten ist • set, Parameter Typ int und Typ Object → Ersetzt Wert an der Stelle int durch Object → Gibt Element zurück, das vorher an der Position war • add, Parameter Typ int und Typ Object → Fügt Object an Stelle int ein → Schiebt alle Elemente ab int um eins nach hinten → Ohne int: an Ende der Liste (add von Collection) • get, Parameter Typ int → Zugriff auf Element in Liste 	7/30
Wildcards	<ul style="list-style-type: none"> • Funktionsweise synchron zu Generics (siehe Generics) 	7/52
Lambda in Liste	<ul style="list-style-type: none"> • Funktionsweise synchron zu Arrays → List<IntPredicate> pred = new ...; → pred.add(n -> n % 2 == 1); 	7/65
Sortieren	<ul style="list-style-type: none"> • mit Comparator (siehe Generics) → Collections.sort(list, new BedingungComparator()); → Sortierlogik durch zweiten Parameter vorgegeben → Comparator Klasse mit selben Typparameter wie list 	7/66
Iterator	<ul style="list-style-type: none"> • Syntax: (Collection<Number> c1 = ArrayList<Number>()); → Iterator<Number> it1 = c1.iterator(); → Gibt Interface Iterator von eigener Iterator Klasse zurück • Verwendung: → while(it1.hasNext()) { it1.next().function(); } • hasNext() → min. 1 Element noch nicht vom Iterator zurückgeliefert • next() → nächstes Element in Collection → bei Collection Reihenfolge unbestimmt → bei Liste Reihenfolge nach Index (ab 0) • Kurzform for-Schleife: → for(String str : coll) {...} → Alle Strings der Collection coll nacheinander 	7/75

	Map	<ul style="list-style-type: none"> • Zwei Parameter: Key und Value <ul style="list-style-type: none"> → Beispiel Map<String, X> = ... ; → Value wird zum jeweiligen Key gespeichert → Abbildung von den Keys ind die Values • Keys müssen immer unterschiedlich sein, Values nicht • Methode put(key,value); <ul style="list-style-type: none"> → fügt Element ein • Methode get: X x1 = map.get(key); <ul style="list-style-type: none"> → Liefert Value zu übergebenen Key zurück → Falls Key nicht vorhanden, return null 	7/90
	Eigene LinkedList	<ul style="list-style-type: none"> • public class ListItem <T> { <ul style="list-style-type: none"> public T key; public ListItem<T> next; } → Verkettung der Elemente durch Attribut der eigenen Klasse • public class MyLinkedList<T> implements List { <ul style="list-style-type: none"> private ListItem<T> head; } → Verweis auf den Kopf der Liste (erstes Element) • Durchlauf durch alle Elemente: <ul style="list-style-type: none"> → Laufpointer: ListItem<T> p = head; (Laufvariable) → Nächster Schritt: p = p.next; → Vorwärtsschritt = Verweis auf nächstes Element → Abbruchkriterium p gleich null, da Liste zu Ende → for(ListItem<T> p = head; p != null; p = p.next) {} • Element hinzufügen: <ul style="list-style-type: none"> • An den Anfang (Index 0): <ul style="list-style-type: none"> → ListItem<T> tmp = new ListItem<T>(); → tmp.key = head; → tmp.next = head; → head = tmp; → Reihenfolge WICHTIG! head ist einziger Verweis auf Liste • Beliebige Stelle (Index n): <ul style="list-style-type: none"> → Arbeiten mithilfe des Laufpointers p → Code erst dann ausführen wenn p an Stelle n-1 → tmp.key = key; → tmp.next = p.next; → p.next = tmp; • Methode add in java.util.List: <ul style="list-style-type: none"> → public void add (int pos, T key) {} → Position und einzufügendes Objekt • Entfernen eines Elements: <ul style="list-style-type: none"> • Am Anfang (Index 0): <ul style="list-style-type: none"> → head = head.next; → Überspringen des erstes Elements • Beliege Stelle (Index n): <ul style="list-style-type: none"> → Analog zum Einfügen → p.next = p.next.next; • Iterator: <ul style="list-style-type: none"> class MyLinkedListIterator<T> implements Iterator <T> { private ListItem<T> p; ... } Iterator intern = Verweis auf Listenelement Wird durch Konstruktor auf Kopf der Liste gesetzt 	

Enums	Syntax	<ul style="list-style-type: none"> • Modifier enum myEnum {} 	1E/6
	Zweck	<ul style="list-style-type: none"> • Zusammenfassung mehrerer Konstanten zu einer Einheit • vordefinierte Menge an Objekten 	1E/7
	Verwendung	<ul style="list-style-type: none"> • myEnum.ENUM; 	1E/9

Exceptions	Laufzeitfehler	<ul style="list-style-type: none"> • werden nicht vom Compiler entdeckt • Abbruch des Programms • Fehlermeldung mit Call-Stack und Zeilennummer • z.B.: Teilen durch 0 	5/2
	Definition	<ul style="list-style-type: none"> • Klasse java.lang.Exception und alle die davon abgeleitet sind 	5/12
	Werfen	<ul style="list-style-type: none"> • Exception wird in Methodenkopf geworfen: <ul style="list-style-type: none"> → my-method (...) throws Exception {...} → auch mehrere Exceptions möglich • Wurf: throw new Exception ("Fehlermeldungstext"); <ul style="list-style-type: none"> → Wirft Exception mit String als Konstruktorparameter • Führt zu Methodenabbruch an der Stelle 	5/13
	Fangen	<ul style="list-style-type: none"> • Falls Methoden potenziell Exceptions werfen könnte: <ul style="list-style-type: none"> • Einfassen in try-Block: try { .. Methode.. } • Bei Exceptionwurf → Abbruch des try-Blocks → Fangen durch catch-Block: catch (Exception name) {...} <ul style="list-style-type: none"> • Kein anderer Code zwischen try und catch Block 	5/24
	Methoden	<ul style="list-style-type: none"> • exc.getMessage() Gibt gespeicherte Fehlermeldung aus • exc.printStackTrace() Gibt Call-Stack aus 	5/33
	Overwrite	<ul style="list-style-type: none"> • In überschreibender Methode Ersetzen der Exception durch abgeleitete Exception erlaubt 	5/44
	Eigene Klassen	<ul style="list-style-type: none"> • public myClass extends Exception {} • Konstruktor: .. super(String);.. 	5/52
	Mehrere Exceptions	<ul style="list-style-type: none"> • throws Excep1, Excep2 {} <ul style="list-style-type: none"> → mehrere catch-Blöcke • Jedoch auch allgemein throws Exception möglich, da Supertyp aller Exceptions <ul style="list-style-type: none"> → selbes Prinzip lässt sich auf catch-Blöcke übertragen → Erster Block in den Exception passt wird verwendet • 2 Exceptions im selben catch-Block: <ul style="list-style-type: none"> → catch (ExcpA ExcpB) {...} 	5/82
	Weiterreichen	<ul style="list-style-type: none"> • Methode mit Exception in anderer Methode • Weiterreichen der Exception an aufrufende Methode: <ul style="list-style-type: none"> → ... throws Exception {} in aufrufender Methode → Dadurch kein try und catch Block für aufgerufene Methode • Spätestens main muss alle weitergereichten Exceptions fangen 	5/93
	try-with-resources	<ul style="list-style-type: none"> • Dokumentation → Parameter bei try-Block 	5/98
	Run-Time-Exceptions	<ul style="list-style-type: none"> • Ausnahme: Diese müssen nicht gefangen werden • Hauptsächlich zur Übersichtlichkeit, da häufige Verwendung • z.B.: IndexOutOfBoundsException <ul style="list-style-type: none"> → Sonst jeder Arrayaufruf in try/catch-Block 	5/99
	Testphase	<ul style="list-style-type: none"> • Throwable und Error <ul style="list-style-type: none"> • Exception und Error von Throwable abgeleitet • Error-Klassen führen immer zum Programmabbruch, werden generell nicht gefangen • Sonderform AssertionError <ul style="list-style-type: none"> • Funktionstest innerhalb Methode • Kurzform: assert Bedingung: "String"; <ul style="list-style-type: none"> → negierte Bedingung in Kurzform! • Warum? Abschaltbar durch Compiler! 	5/117

	JUnit-Tests	<ul style="list-style-type: none"> • Zweck: Funktionstest der Methode als Ganzes • Imports: <pre>import static org.junit.Assert.assertEquals; import static org.junit.Assert.assertTrue; import static org.junit.jupiter.api.Assertions.assertThrows; import org.junit.jupiter.api.Test; import org.junit.jupiter.api.BeforeEach;</pre> • Verwendung: <ul style="list-style-type: none"> • Verwenden eigener Quelldatei und Import der Testdatei • @Test führt zur direkten Ausführung der Methode • @BeforeEach Ausführung dieser Methode vor @Test Methoden • Erstellung von void Methoden zum Testen • Funktionen: <ul style="list-style-type: none"> • assertEquals(param1, param2); → Test auf Gleichheit von 2 Parametern → Mit 3 Parametern: 3. Parameter = Intervall für Abweichung • assertEquals(...); • Objektidentität • assertThrows(...); • Überprüft ob erwarteter Error geworfen wird 	5/132
--	-------------	--	-------

Fehler	Compiler	<ul style="list-style-type: none"> • werden vom Compiler erkannt • z.B.: falsche Klammerung, fehlendes ;, etc 	3A/3
	Runtime	<ul style="list-style-type: none"> • werden während Laufzeit erkannt • z.B.: Division durch 0, Array out of bounds, etc. 	3A/4

Files	Wissenswertes	<ul style="list-style-type: none"> • Package java.nio.file • Sammlung von Klassenmethoden 	8/184
	Methoden	<ul style="list-style-type: none"> • Path path = Paths.get(...); • if(Files.exists(path){} (True, falls dort etwas existiert) • if(Files.isReadable(path){} (True, falls Leserechte vorhanden) • if(Files.isWritable(path){} (True, falls Schreibrechte vorhanden) • if(Files.isRegularFile(path){} (True, falls Typ der Datei regulär ist) • if(Files.isDirectory(path){} (True, falls Typ Verzeichnis) • long size = File.size(path); (Ruft Länge der Datei ab) 	8/186
	Manipulation	<ul style="list-style-type: none"> • Files.createFile(path) (Erzeugt Datei an Pfadstelle) • Files.copy(path1,path2); (Kopiert Datei an Pfad 1 nach Pfad 2) • Files.move(path3, path4); (Bewegt bzw benennt die Dateien um) • Files.delete(path); (Entfernt das Objekt an der Pfadstelle) → NoSuchFileException bei Nichtexistieren des Objekts • Files.deleteIfExists(path); (Falls nicht vorhanden, passiert nichts) 	8/194

GUI	Window Manager	<ul style="list-style-type: none"> • Systemprozess → Wird beim Booten gestartet • Läuft als Service im Hintergrund • Stellt generelle Funktionen zur Verfügung 	10/2
-----	----------------	---	------

Frame	<ul style="list-style-type: none"> • Window mit Rahmen (Subtyp von Klasse Window) • Teil des Packages java.awt (abstract window toolkit) • Standardmäßig unsichtbar • Erzeugung: <ul style="list-style-type: none"> → <code>Frame frame = new Frame(new String("Hello World!"));</code> → Ein Konstruktor z.B. mit String → Titel des Rahmens • Umgang: <ul style="list-style-type: none"> → Erzeugung eines Subtypes von Frame um Sachen hinzuzufügen → ActionListener werden oft in SubFrame geschrieben (private class) • Methoden: <ul style="list-style-type: none"> → <code>frame.setVisible(true);</code> (Fenster sichtbar) → <code>frame.setVisible(false);</code> (Fenster unsichtbar) → <code>frame.setBackground(new Color(0, 0, 0));</code> (setzt Farbe) → <code>frame.dispose();</code> (alle Ressourcen werden freigegeben) → <code>frame.setExtendedState(Frame.ICONIFIED);</code> (ikonifiziert) → <code>frame.setExtendedState(Frame.NORMAL);</code> (deikonifiziert) → <code>frame.setExtendedState(Frame.MAXIMIZED_HORIZ);</code> → Setzte Framebreite auf Bildschirmbreite → <code>frame.add(...);</code> (Hinzufügen von z.B. Buttons) 	10/11
Buttons	<ul style="list-style-type: none"> • Erzeugung: <ul style="list-style-type: none"> → <code>Button button1 = new Button("String");</code> → String = Text auf Button → Größe des Buttons wird durch String determiniert • Werden mithilfe von <code>frame.add(button1);</code> ins Frame eingefügt • Methoden: <ul style="list-style-type: none"> → <code>button.setFont(new Font("font", Font.BOLD, size));</code> → <code>button.addActionListener(actionListenerClass);</code> → <code>button.setLabel("String");</code> (ändert Titel) 	10/25
Action-Listener	<ul style="list-style-type: none"> • Klasse die ActionListener (java.awt.event) implementiert benötigt • funktionale Methode: <code>void actionPerformed (ActionEvent event) {}</code> • Dann Hinzufügen zum Button: <ul style="list-style-type: none"> → <code>button.addActionListener(actionListenerClass);</code> → Verwendung eines Lambda-Ausdrucks hier auch möglich (Folie 69) → Automatische Erzeugung eines Threads, der auf Eingaben wartet → Event Dispatch Thread 	10/33
ActionEvent	<ul style="list-style-type: none"> • Hier Verwendung als Parameter für <code>actionPerformed(ActionEvent event)</code> • Methoden: <ul style="list-style-type: none"> → <code>event.getWhen();</code> (Rückgabe des Zeitpunkt als long in ms seit 1970) → Hilfreiche Umrechnung dieser mithilfe der Klasse Timestamp → Methoden von Timestamp: <code>getHour(), getMinute()</code> 	10/50
Weitere Listener und Events	<ul style="list-style-type: none"> • Aufbau: Listener / Event • KeyListener / KeyEvent • MouseListener / MouseEvent • MouseMotionListener / MouseEvent • MouseWheelListener / MouseWheelEvent • WindowFocusListener / WindowEvent • WindowListener / WindowEvent • WindowStateListener / WindowEvent • <code>frame.addKeyListener(new MyKeyListener(...));</code> → analog für Rest 	10/70
KeyListener	<ul style="list-style-type: none"> • <code>public interface KeyListener {</code> <ul style="list-style-type: none"> <code>public void keyPressed (KeyEvent event);</code> <code>public void keyReleased (KeyEvent event);</code> <code>public void keyTyped (KeyEvent event);</code> • Jede dieser Methoden ähnlich wie <code>actionPerformed</code> • Zu jedem nicht functional Listener-Interface gibt es eine zugehörige Adapter-Klasse <ul style="list-style-type: none"> → Damit nicht alle Methoden implementiert werden müssen → Sondern hier z.B. mithilfe von "extends KeyAdapter" nur <code>keyPressed</code> 	10/77

Mouse Listener	<ul style="list-style-type: none"> • public interface MouseListener { public void mouseClicked(MouseEvent event); public void mousePressed(MouseEvent event); public void mouseReleased(MouseEvent event); public void mouseEntered(MouseEvent event); public void mouseExited(MouseEvent event); } • public interface MouseMotionListener { public void mouseDragged(MouseEvent event); public void mouseMoved(MouseEvent event); } • public interface MouseWheelListener { public void mouseWheelMoved(MouseWheelEvent event); } 	10/88
Window Listener	<ul style="list-style-type: none"> • public interface WindowListener { public void windowOpened(WindowEvent event); public void windowClosing(WindowEvent event); public void windowClosed(WindowEvent event); public void windowActivated(WindowEvent event); public void windowDeactivated(WindowEvent event); public void windowIconified(WindowEvent event); public void windowDeiconified(WindowEvent event); } • public interface WindowStateListener { public void windowStateChanged (WindowEvent event); } • public interface WindowFocusListener { public void windowGainedFocus(WindowEvent event); public void windowLostFocus(WindowEvent event); } 	10/98
Adapter	<ul style="list-style-type: none"> • Verbindung zwischen Interfaces und der Klasse Adapter: • abstract public class KeyAdapter implements KeyListener • abstract public class MouseAdapter implements MouseListener, MouseMotionListener, MouseWheelListener • abstract public class WindowAdapter implements WindowListener, WindowFocusListener, WindowStateListener 	10/101
Canvas	<ul style="list-style-type: none"> • abgegrenzte Zeichenfläche in einem Fenster • public class MyCanvas extends Canvas { ... } → Eigener Subtyp um Methode paint selbst zu implementieren • Zeichenbeispiel mit Code ab 10/112 	10/107
Checkbox	<ul style="list-style-type: none"> • Besteht aus kleinem Button und kurzem Text • Interface ItemListener mit funktionaler Methode: → public void itemStateChanged(ItemEvent event); • checkbox.isSelected(): True, falls checkbox an ist 	10/132
Choice	<ul style="list-style-type: none"> • Repräsentiert ein Auswahlménü • Choice choice = new Choice(); (am Anfang leer) • choice.add("Yes"); choice.add("No"); • choice.select(int); (Zur Anfangswahl) • Greift auch auf ItemListener zurück • choice.getSelectedItem() (Wird benötigt für Listener) → Zugriff auf ausgewähltes Item 	10/146
Label	<ul style="list-style-type: none"> • Rechteck mit Text / Nicht vom User interagierbar • Label label = new Label("String"); • label.setAlignment(Label.CENTER); (LEFT,RIGHT) • label.setBackground(Color); • Benötigt anderen Komponenten für Veränderungen, z.B. Button 	10/154
List	<ul style="list-style-type: none"> • Diese List in java.awt, andere List (Generics) in java.util • Menü mit Auswahlpunkten • List list = new List(int, boolean); → int gibt an, wieviele Menüpunkte gleichzeitig angezeigt werden → Mehr Objekte als Int → Scrollbar → boolean gibt an, ob mehrere Menüpunkte auswählbar sind • int[] selectedIndexes = list.getSelectedIndexes; → Speichert Positionen der Punkte, die ausgewählt sind 	10/167

Scrollbar	<ul style="list-style-type: none"> • Scrollbar bar = new Scrollbar(...,...,...); • 4 Parameter: <ul style="list-style-type: none"> → Scrollbar.VERTICAL oder Scrollbar.HORIZONTAL → Startwert als int → Minimalwert z.B. 0 (keinen Einfluss auf Größe) → Maximalwert z.B. 255 (keinen Einfluss auf Größe) • Funktionales Interface AdjustmentListener mit Methode: <ul style="list-style-type: none"> → public void adjustmentValueChanged (AdjustmentEvent event); • Methode: event.getValue(); (Liefert Wert der Scrollbar) 	10/172
Text-Component	<ul style="list-style-type: none"> • zwei Subtypen TextField und TextArea • TextField: <ul style="list-style-type: none"> → Eingabezeile für den Nutzer (Passworteingabe) → TextField field = new TextField(maxLength); → field.setEchoChar('*'); (Ersetzen der Zeichen) → Reset mithilfe von field.setEchoChar(0); → nutzt KeyListener bzw KeyAdapter → Relevant: event.getKeyChar() • TextArea: <ul style="list-style-type: none"> → Bereich mit beliebig vielen Eingabezeilen → TextArea area = new TextArea("String", int1, int2, scrollbars); → int1: Anzahl Zeilen int2: Länge Zeilen → scrollbars: TextArea.SCROLLBARS_BOTH für beide Scrollbars → VERTICAL_ONLY HORIZONTAL_ONLY NONE BOTH → nutzt FocusListener mit focusGained und focusLost 	10/185
Hierarchie	<ul style="list-style-type: none"> • Von Component abgeleitet: <ul style="list-style-type: none"> → Button Canvas Checkbox Choice → Label List Scrollbar TextComponent → Container Von Container abgeleitet: <ul style="list-style-type: none"> → Window Von Window abgeleitet: <ul style="list-style-type: none"> → Frame 	10/208
Container	<ul style="list-style-type: none"> • fasst mehrere Komponenten zu einer zusammen • Wichtig: Kann auch andere Container enthalten • Methoden: <ul style="list-style-type: none"> → void paint (Graphics graphics) → void add (component) (wie bisher) → void add(component, Object constraints) (dazu gleich mehr) → void setLayout (LayoutManager manager) (Layout, nutzt constraints) → void validate() (Aktualisierung) 	10/216
Layout-Manager	<ul style="list-style-type: none"> • Automatische Erstellung bei Erstellung eines Containers/Subtyps <ul style="list-style-type: none"> → Abhängig vom Typ Window und Frame: BorderLayout • BorderLayout: <ul style="list-style-type: none"> → Findet seine Verwendung in add-Methode mit constraints → z.B. BorderLayout.NORTH (.CENTER,.WEST,.EAST,.SOUTH) → Automatische Unterteilung des Frames in 5 Bereiche → bei normaler add-Methode wird immer .CENTER angewandt • Andere LayoutManager: <ul style="list-style-type: none"> → BoxLayout: Anlegung der Komponenten in Reihe → GridLayout: Matrixartige Anlegung → FlowLayout: Automatische Größenanpassung in Zeilen → CardLayout: Anzeigung der Komponenten nacheinander • frame.validate(): <ul style="list-style-type: none"> → MUSS nach Größen-/Anzahländerungen aufgerufen werden → richtet Layout wieder richtig aus 	10/226

	Java Swing	<ul style="list-style-type: none"> • Selbe Funktionen wie java.awt, jedoch mehr Funktionen • Package javax.swing enthält: <ul style="list-style-type: none"> → JFrame extends java.awt.Frame → JComponent extends Java.awt.Container → von JComponent abgeleitet: <ul style="list-style-type: none"> → JButton JCheckBox JLabel JList<T> → JScrollBar JTextComponent(JTextArea JTextField) → JButton z.B. indirekt über AbstractButton von JComponent • Tooltips: (MouseOver-Text) <ul style="list-style-type: none"> → button.setToolTipText("String"); • Randdarstellungen: <ul style="list-style-type: none"> → .setBorder(Border border); → Nutzung von Klasse BorderFactory → Bsp: BorderFactory.createLineBorder(Color.RED, thickness); → BorderFactory.createEmptyBorder(); • Look and Feel: Plattformunabhängig gleiches Aussehen <ul style="list-style-type: none"> → Folien 10/260-267 • Key Bindings: <ul style="list-style-type: none"> → sehr codelastig, deswegen Folie zum Verständnis → 10/268-276 • Drag&Drop • Assistive Technologies • Separierung Hauptmenü und eigentliches Fenster • Weitere Swing-Klassen: <ul style="list-style-type: none"> → JFormattedTextField (einzugebender Text vorformatiert) → JPasswordField → JRadioButton (kleiner anklickbarer Kreis) → JToolBar → JSlider (Schieberegler) → Popup → JTable (Tabelle) <ul style="list-style-type: none"> → JTable table = new JTable (Object[][], Object[]) → Erstes Array Matrix Zweites Array Namen der Spalten → JScrollPane scrollPane = new JScrollPane(table); → Einkapselung des Objekts Zeigt nur Ausschnitt mit Scrollbars → table.setFillViewportHeight(true); → Streckung der Tabelle auf die vertikal benötigte Höhe → int[] selectedRows = table.getSelectedRows(); → Abfragen der momentan angeklickten Zeilen 	10/240
--	------------	--	--------

if-Anweisung	Syntax	<ul style="list-style-type: none"> • if (Bedingung) {} ; • bei Einzelner Anweisung auch ohne {} 	1B/49
	Funktion	<ul style="list-style-type: none"> • Verzweigung des Programms an gewissen Stellen 	1B/49
	else	<ul style="list-style-type: none"> • Syntax: else {} • Fängt alle Fälle ab, die nicht bei if auftreten 	1B/54
	weitere Verzweigung	<ul style="list-style-type: none"> • switch (Typ) { <ul style="list-style-type: none"> case myTyp: ... break; default: ... ; } • myTyp keine Variable, muss zur Laufzeit schon feststehen 	3C/218

Generics	Wrapper-Klassen	<ul style="list-style-type: none"> • Zu jedem primitiven Datentypen eine Wrapper-Klasse • Selber Name mit großem Anfangsbuchstaben <ul style="list-style-type: none"> • Außer int und char → Integer und Character • Konstruktor mit einzelner Parameter seines Datentyps • z.B.: in = new Integer; • Zugriff mit z.B.: in.intValue(); • Boxing / Unboxing: <ul style="list-style-type: none"> • Weitgehend austauschbar → automatische Umwandlung 	6/1
----------	-----------------	--	-----

Klassen	<ul style="list-style-type: none"> • Beispiel: <code>public class Pair <T1, T2 > {...}</code> <ul style="list-style-type: none"> → Klasse Pair ist generisch bzw. mit T1 und T2 parametrisiert → T1 und T2 sind Typparameter der Klasse Pair • Konstruktor: z.B.: <ul style="list-style-type: none"> → <code>Pair <Integer,Double> p = new Pair<Integer,Double>(i, d);</code> → Instanziierung von Pair mit Integer und Double 	6/12
Typparameter	<ul style="list-style-type: none"> • Verwendung bei Attributen: <ul style="list-style-type: none"> → z.B.: <code>private T1 attribute1;</code> • Verwendung bei Methoden: <ul style="list-style-type: none"> → als Rückgabotyp → als aktueller Parameter (auch im Konstruktor) → nur bei Objektmethoden • Mögliche Parameter: <ul style="list-style-type: none"> → Eigene Klassen → Arrays (Primitiv/Referenz) → Interfaces (eingeschränkt → später) → Parametrisierte Klassen und Interfaces 	6/15
Methoden	<ul style="list-style-type: none"> • In nicht generische Klasse: <ul style="list-style-type: none"> → Parametrisierung vor Rückgabotyp → z.B.: <code>public <T1,T2> Pair<T1,T2> myMethod() ...</code> • in generischer Klasse: <ul style="list-style-type: none"> → param. Typparameter müssen nicht mehr angegeben werden 	6/38
Vererbung	<ul style="list-style-type: none"> • z.B.: <code>void m (Pair<X,Y> paar) ...</code> <ul style="list-style-type: none"> → keine generische Methode, da Parameter fest → X oder Y nicht durch abgeleitete Klassen ersetzbar 	6/59
Abkürzung	<ul style="list-style-type: none"> • <code>Pair<String,Integer> pair</code> • <code>pair = new Pair<> ("Hello", 123);</code> → Weglassen der Typparameter bei Initialisierung 	6/65
Eingeschränkte Typparameter	<ul style="list-style-type: none"> • <code>public class A <T extends X></code> → beschränkt auf X oder Subtyp • Methoden aus X dann auch in der Klasse A verwendbar • z.B.: <code><T extends Number></code> für Zahlen (kein Character) • mehrfach eingeschränkt: (auch Interfaces) <ul style="list-style-type: none"> → <code><T extends X & Int1 & Int2></code> → Klasse muss als Erstes kommen 	6/72
Wildcards	<ul style="list-style-type: none"> • Einschränkungen auch bei nichtgenerischen Methoden • Aktueller Parameter: <ul style="list-style-type: none"> • Upper-Bounded: <ul style="list-style-type: none"> → <code>double m (X<? extends Number> n) ...</code> → Typ des Parameters n vom Typ Number oder Subtyp • Lower-Bounded: <ul style="list-style-type: none"> → <code>double m (X<? super Double> d) ...</code> → Typ des Parameters n vom Typ Double, Basisklasse von Double, allen von Double implementierten Interfaces → Hier als Beispiel: Nur Double, Number und Object • Unterschied: <ul style="list-style-type: none"> • generische Methode: <ul style="list-style-type: none"> → <code>public <T extends Number> double m (X<T> n) ...</code> • nichtgenerische Methode: <ul style="list-style-type: none"> → <code>public double m (X<? extends Number> n) ...</code> • nur <code>(X<?> x)</code> → allgemeinster Fall (extends Object) • Verwendung: <ul style="list-style-type: none"> → In-Parameter: extends → Out-Parameter: super → In/Out-Parameter: weder noch → Rückgabe: weder noch 	6/86

	Comparator	<ul style="list-style-type: none"> • Methode compare mit 2 Typparametern und Rückgabebetyp int → public int compare (T t1, T t2) ... ⇒ in Klasse z.B. <T extends Number> • Rückgabewert int: → -1, falls erster Parameter dem Zweiten vorangehend → +1, falls zweiter Parameter dem Ersten vorangehend → 0, falls beide äquivalent • Funktion: → Erlaubt den Vergleich von Elementen auf verschiedenste Art • Beispiel mit Referenztypen: 6/120 	6/105
	Einschränkungen	<ul style="list-style-type: none"> • Keine primitiven Datentypen als Instanziierungen • Keine Erzeugung von Objekten / Arrays von Typparametern • Keine Klassenattribute von Typparametern • Kein Downcast oder instanceof mit Typparametern • Kein throw-catch mit Typparametern • Keine MethodenÜberladung auf Typparametern 	6/140

Graphic Fallbeispiel	Applets	<ul style="list-style-type: none"> • grafikorientiertes Programm (hier in HTML) • Syntax: public class myApplet extends Applet {} 	2/11
	Graphics	•	2/14
	GeomShape2D	•	2/38
	Beachte	• Folien nachlesen, zu spezifisch	2

Interfaces	Syntax	• public interface myInterface {}	1G/68
	Methoden	<ul style="list-style-type: none"> • Rückgabebetyp myMethod(); (nicht implementiert, nur definiert) → dürfen aber auch implementiert werden → automatisch public • Objektmethoden: default myMethod(); 	1G/69
	Attribute	• Möglich, allerdings nur als Klassenkonstanten (public final)	4C/38
	Implementierung	• public myClass implements MyInterface1,MyInterface2 {}	1G/72
	Zweck	• Auslagerung von häufigem Code in Interface	1G/65
	Unterschied Abstrakte Klassen	<ul style="list-style-type: none"> • Interfaces können Mehrfachvererbung • Abstrakte Klassen können: <ul style="list-style-type: none"> • von Klassen abgeleitet werden • Methoden und Attribute, die nicht public sind 	4C/47
	Funktionale Interfaces	<ul style="list-style-type: none"> • Interfaces mit genau einer funktionalen Methode → Methode, die weder default oder static ist 	4C/52

Keywords	Verwendung	• Kann nur an bestimmten Stellen stehen (for, int,while,..)	1A/147
----------	------------	---	--------

Klassen	Kopf	• Modifier class myClass {}	1E/21
	Attribute	• Eigenschaften der Objekte von myClass	1E/24
	Methoden	• TBD (siehe Methoden)	
	Konstruktor	<ul style="list-style-type: none"> • Einrichtung von neuen Objekten von myClass → Initialisierung der Attribute • Syntax: Modifier myClass (Parameter) {}; 	1E/115
	String	• Aufrufe: (.length, .charAt, .indexOf, .matches)	3B/75

	Klassenobjekte	<ul style="list-style-type: none"> • Syntax: Modifier static ... • Klassenattribut → Zugriff auf immer selbe Speicherstelle • auch über myClass.myStaticVariable ansprechbar 	3B/169
	java.lang.Object	<ul style="list-style-type: none"> • kein extends → Object als Basisklasse 	3B/198
	Verborgene Informationen	<ul style="list-style-type: none"> • anonymes Objekt: <ul style="list-style-type: none"> • Informationen zur Verwendung der Klasse • Methodentabelle: Erstellung zur Laufzeit (3C/35) 	3C/18
	Klassenmethoden	<ul style="list-style-type: none"> • Zugriff auf Klassenmethoden/attribute → jedoch nicht auf Objektmethoden/attribute • Implementation vom statischen Typ abhängig (3C/113) 	3C/52

Kommentare	Einzeilig	<ul style="list-style-type: none"> • // 	1A/186
	Bereich	<ul style="list-style-type: none"> • /* - */ 	1A/187
	Javadoc	<ul style="list-style-type: none"> • /** - */ (nach /** Enter) (3C/90) • Tags @ und neue Zeilen mit * @param für Parameter @throws für Fehlermeldungen @return Für Rückgabewerte 	1A/188

Konventionen	Identifizier	<ul style="list-style-type: none"> • Methoden/Variablen: myMethod/myVariable • Klassen: MyClass • Konstanten: MY_CONSTANT (_ statt Leerzeichen) • Package: mypackage → Firmen: umgedrehter Domain Name mit _ • Erstes Zeichen darf keine Ziffer sein, keine Keywords 	1A/150
--------------	--------------	---	--------

Konversionen	implizite	<ul style="list-style-type: none"> • Von größerem Datentyp in kleineren → ohne Probleme 	1B/211
	explizite	<ul style="list-style-type: none"> • Zieltyp in Klammern angeben (z.B.: i = (int)myLong) 	1B/214

Lambda-Ausdrücke	Syntax	<ul style="list-style-type: none"> • myFunctionalInterface myVariable = ... -> ... • Einrichten unsichtbarer Klasse, die das Interface implementiert 	4C/59
	Funktion	<ul style="list-style-type: none"> • Abgekürzte Schreibweise für den Aufruf einer Hauptmethode eines funktionalen Interfaces • Bilden der funktionalen Methode: → Parameter -> Methode • Allgemeine Form: → (int n, double d)-> {return ...} → Kurzform nur in einfachen Fällen möglich 	4C/63
	Closure	<ul style="list-style-type: none"> • Information aus Entstehungskontext mitgespeichert (Variable) → Bei Verwendung des Lambda-Ausdrucks mitverwendet 	4C/67
	Definition	<ul style="list-style-type: none"> • Literale von Funktionstypen 	4C/70
	Fallbeispiel Prädikate	<ul style="list-style-type: none"> • Beispiel zur Verdeutlichung 	4C/71

Methoden	Syntax	<ul style="list-style-type: none"> • Modifier Rückgabotyp myMethod (Parameterliste) {}; • Signatur(unique): Name + Parameterliste 	3C/75
	Methodenaufruf	<ul style="list-style-type: none"> • myObject.myMethod(Parameterliste); 	1A/33
	Rückgabotypen	<ul style="list-style-type: none"> • void → Liefert keinen Wert zurück • !void → zwingend return ...; 	1E/32

	Parameter	<ul style="list-style-type: none"> • in Parameterliste: (Datentyp myVariable,...) • mit this.myVariable wird das Attribut angesprochen • formale Parameter: <ul style="list-style-type: none"> • alle Parameter bei der Methodendefinition • aktuelle Parameter: <ul style="list-style-type: none"> • alle Parameter beim Methodenaufruf • Nicht selber Datentyp benötigt, solange implizit konvertierbar 	1E/107
	Abstrakt	<ul style="list-style-type: none"> • Syntax: abstract Modifier myMethod() • Erzwingt abstrakte Klasse • 'Rahmen' für andere Klassen 	
	Variable Parameterzahl	<ul style="list-style-type: none"> • ... myMethod (double... d){} • Entweder Übergabe eines Arrays oder viele double Werte → Umwandlung in Array vom Typ Double 	3C/65
	Überschreiben	<ul style="list-style-type: none"> • Eigenschaften: <ul style="list-style-type: none"> • Signatur zwingend gleich • private → nichts → protected → public <ul style="list-style-type: none"> • Zugriff von Basis zu Subtyp erweitert (möglich) • Bei Rückgabe von Referenz → durch Subtyp ersetzbar → auch bei Exceptions 	3C/95
	Überladen	<ul style="list-style-type: none"> • Selber Name, andere Parameterliste (→ unterschiedl. Signatur) • Modifier und Rückgabetyt können variieren 	3C/115

Modifier	Packages	<ul style="list-style-type: none"> • package myPackage; (oberster Befehl klein) • import myPackage.*; (* → keine Subpackages) • Namenskonflikte → mypackage.myClass ... (Qualifizierung) 	1F/93
	Zugriff	<ul style="list-style-type: none"> • private: nur in Klasse selbst • nichts: private + package • protected: nichts + Vererbung • public: protected + alle Imports • Nur eine public myClass oder myEnum pro Quelldatei 	1F/100

Nested classes	Wissenswertes	<ul style="list-style-type: none"> • Verschaltelte Klassen → eine Klasse in der anderen eingebettet • Äußere und innere Klasse → public oder package Klasse = Äußere Klasse → public, private oder protected = innere Klasse → Nur eine Klasse kann public sein! → Name der Quelldatei 	9/2
	Zugriff	<ul style="list-style-type: none"> • OuterClass.InnerClass → nur möglich, falls innere Klasse public • Objekt von Y benötigt ein Objekt von X zum Bezug → Attribut der äußeren Klasse vom Typ der inneren Klasse benötigt 	9/11

Objekte	Erzeugung	<ul style="list-style-type: none"> • Klasse myClass = new Klasse (Parameterliste); 	1A/147
---------	-----------	---	--------

Optional	Eigenschaften	<ul style="list-style-type: none"> • Hilfsklasse aus dem Package java.lang • Erzeugtes Objekt kapselt ein Objekt seines Typparameters ein • bequeme Möglichkeit um mit Referenz null umzugehen 	8/3
----------	---------------	---	-----

	Methoden	<ul style="list-style-type: none"> • Erzeugung: <ul style="list-style-type: none"> → <code>Optional<Number> opt1 = Optional.OfNullable(new Integer(123));</code> → <code>Optional<Number> opt1 = Optional.OfNullable(null);</code> (leer") • Abruf: <ul style="list-style-type: none"> → <code>OptionalObject.get();</code> → Bei null: <code>NoSuchElementException</code> • <code>orElseGet</code>: <ul style="list-style-type: none"> → Wenn Object leer, Zurücklieferung eines anderes Wertes → <code>opt1.orElseGet(() -> 0);</code> → Gibt 0 zurück, falls Object == null • <code>isPresent</code>: <ul style="list-style-type: none"> → Wenn Object nicht null, Ausführung des Lambdas • <code>map</code>: <ul style="list-style-type: none"> → <code>Optional<Number> opt3 = opt1.map(x -> x * x);</code> → <code>opt3</code> verweist auf Optional-Object, das ein Number-Objekt einkapselt mit dem Quadrat von <code>opt1</code> → Falls Object null auch Container null • <code>filter</code>: <ul style="list-style-type: none"> → Liefert Optional-Object vom selben generischen Typ zurück → Wenn Lambda-Ausdruck true, dann speicher des Verweises → Wenn Lambda-Ausdruck false, dann speicher von null → <code>Optional<Number> opt4 = opt3.filter(x -> x%2 == 1);</code> 	8/4
--	----------	---	-----

Primitive Datentypen	Ganzzahlig	<ul style="list-style-type: none"> • byte: 8 Bits • short: 16 Bits • int: 32 Bits • long: 64 Bits • <code>Integer.MAX_VALUE</code> / <code>Integer.MIN_VALUE</code> etc. 	1B/120
	Gebrochen	<ul style="list-style-type: none"> • float: 32 Bits • double: 64 Bits (Genauigkeit 500 Millionen mal höher) • Vergleich von gebrochenen Zahlen mit maximalen Fehler • <code>Double.MAX_VALUE</code> / <code>Double.POSITIVE_INFINITY</code> 	1B/134
	Literale	<ul style="list-style-type: none"> • wörtlich hingeschriebene Werte, automatisch int <ul style="list-style-type: none"> • Typ long: 123L • Typ byte / short: automatisch falls klein genug • gebroche Literale automatisch double <ul style="list-style-type: none"> • Typ float: 12.34F • Exponenten 1.2E34 ($1,2 * 10^{34}$) 	1B/125
	Logiktyp	<ul style="list-style-type: none"> • boolean: binär (true oder false → booleschen Literale) • Operationen: <ul style="list-style-type: none"> • Negation: <code>!a</code> • Und: <code>a&&b</code> • Oder: <code>a b</code> 	1B/153
	Zeichen	<ul style="list-style-type: none"> • char: Schriftzeichen in Einzelhochkommas ('a') <ul style="list-style-type: none"> • Unicode: Kodierung als Zahl mit 16 Bit <ul style="list-style-type: none"> → ASCII: Zeichen 0-127 ISO-Latin-1: Zeichen 127-255 • Unicode-Nummer auch hexadezimal '\u....' 	
	Overflow	<ul style="list-style-type: none"> • Falls Wert zu groß für Datentyp • Keine Fehlermeldung, Informationsverlust! 	1B/129
	Nullwert	<ul style="list-style-type: none"> • Falls nicht initialisiert, nur definiert: <ul style="list-style-type: none"> • Zahlentypen: 0 • boolean: false • Referenzen: null 	3C/154

Programm-ablauf	Programm	<ul style="list-style-type: none"> • Sequenz von Informationen (Quelltext und Java-Bytecode) 	1A/61
	Prozesse	<ul style="list-style-type: none"> • Werden nacheinander von CPU abgearbeitet (Warteschlange) • → Illusion von Multitasking 	1A/61
	Anweisung	<ul style="list-style-type: none"> • kleinste Einheit / der Reihe nach ausgeführt 	1A/72

Referenztypen	Definition	<ul style="list-style-type: none"> • !(Primitive Datentypen) 	3B/2
	Aufzählung	<ul style="list-style-type: none"> • Klassen, Arrays, Interfaces, Enum 	3B/3
	Enum	<ul style="list-style-type: none"> • Begründung: Sind auch Klassen, deswegen Referenzen 	3B/14
	Gleichheit	<ul style="list-style-type: none"> • Test auf Gleichheit problematisch, da Referenzen → Objektidentität vs. Wertgleichheit → deep vs shallow copy 	3B/49
	statisch vs dynamisch	<ul style="list-style-type: none"> • Syntax: static myClass = new dynamic(); • statisch: <ul style="list-style-type: none"> • unveränderlich mit Referenz verknüpft (darüber definiert) • entscheidet auf welche Methoden zugegriffen werden darf • dynamisch: Typ des Objekts der Referenz <ul style="list-style-type: none"> • muss gleich oder Subtyp des statischen Typs sein • entscheidet welche Implementation der Methode 	3B/152

Schleifen	for-Schleife	<ul style="list-style-type: none"> • for (Ausführung davor; Bedingung; Ausführung danach) {}; • Beispiel: for(int i = 0; i < x; i++ {}; • bei Einzelner Anweisung auch ohne {} • auch for(;;) möglich → Endlosschleife 	1A/130
	while-Schleife	<ul style="list-style-type: none"> • while (Bedingung) {}; • do {} while(Bedingung); (fußgesteuert) • bei Einzelner Anweisung auch ohne {} 	1A/123
	break;	<ul style="list-style-type: none"> • Verlassen der nächsthöheren (inneren) Schleife • Fortfahren mit nächster Anweisung nach Schleife 	1B/50
	continue;	<ul style="list-style-type: none"> • Beendet momentanen Schleifendurchlauf • Als Nächstes wieder Prüfen der Bedingung 	1B/51

Scope	Definition	<ul style="list-style-type: none"> • Gültigkeitsbereich von Identifiern 	3A/47
	Modifier	<ul style="list-style-type: none"> • Klassen, Methoden, Variablen etc. → siehe oben 	3A/49
	lokale Variablen	<ul style="list-style-type: none"> • Innerhalb von Methodenrumpfen, Schleifen, etc. 	3A/52
	this	<ul style="list-style-type: none"> • Bei gleichen Namen von lokalen Variablen und Attribute → this 	3A/56

Semikolon	Verwendung	<ul style="list-style-type: none"> • Semikolon nach jeder abgeschlossenen Anweisung (;) 	1A/22
-----------	------------	--	-------

Speichermodell	Abstraktion	<ul style="list-style-type: none"> • großes Feld von Maschinenwörtern (Länge immer gleich, aber abhängig von der Hardware (32bit / 64bit)) 	1A/24
	Speichernutzung	<ul style="list-style-type: none"> • Name eines Objekts/Arrays wird als Referenz auf Speicherort abgelegt • → deswegen Operator new" 	1A/28
	Primitive Datentypen	<ul style="list-style-type: none"> • Name verweist tatsächlich auf konkrete Speicherstelle 	1A/139
	Program Counter	<ul style="list-style-type: none"> • enthält Adresse der nächst auszuführenden Anweisung 	1A/143
	Methodenaufruf	<ul style="list-style-type: none"> • Einrichten des Stackpointers Callstack Frame 	1E/51
	Referenzen vs Objekte	<ul style="list-style-type: none"> • Unterschiede im Speicher, siehe Folien 	3B/20

	Garbage Collector	<ul style="list-style-type: none"> • Teil des Laufzeitsystems, löscht unreferenzierte Objekte 	3B/210
--	-------------------	--	--------

Streams	Eigenschaften	<ul style="list-style-type: none"> • Klasse Optional fürs Verständnis relevant! (weiter oben) • generisches Interface im Package java.util.stream • einheitliche Schnittstelle für Listen, Arrays, Dateien,... • von potentiell unendlicher Länge 	8/19
	Stream aus Listen	<ul style="list-style-type: none"> • <code>Stream<Number> stream1 = list.stream();</code> • <code>Stream<Number> stream2 = stream1.filter(myPredicate);</code> → Neuer Stream mit gefilterten Objekten abh. vom Prädikat • <code>Stream<Number> stream3 = stream1.map(myFunction);</code> → Neuer Stream mit angepassten Objekten abh. von der Funktion • <code>Optional<Number> opt = stream3.max(new Comparator());</code> → Gibt abhängig vom übergebenen Comparator ein Element zurück → Da der Stream auch leer sein kann, Objekt vom Typ Optional • Funktioniert genauso mit Arrays allerdings Parameter bei <code>Arrays.stream</code> mit Arraytyp 	8/21
	Erzeugung	<ul style="list-style-type: none"> • <code>Stream<Number> stream1 = Stream<Number>.of(new Integer(1), new Integer(2));</code> 	8/41
	Iteration	<ul style="list-style-type: none"> • <code>Iteration iter = stream.iterator();</code> • <code>while (iter.hasNext()) { Number n = iter.next(); ..code.. }</code> 	8/48
	Liste aus Stream	<ul style="list-style-type: none"> • <code>List<String> list = stream.collect(Collectors.toList());</code> → <code>Collectors.toList()</code> liefert generisches Interface <code>Collector</code> zurück • <code>Number[] a = stream.toArray(Number[]::new);</code> → Methodennamen als Lambda-Ausdrücke (<code>Number[]::new</code>): <ul style="list-style-type: none"> • Fachbegriff <code>method reference</code> • Folien 8/55 - 8/83 lesen Dokumentation 	8/49
	Streams und Dateien	<ul style="list-style-type: none"> • Erstellen eines Pfades: <code>Path path = Paths.get(..., ..., ...);</code> → <code>Path</code> und <code>Paths</code> in der Package <code>java.nio.file</code> → <code>Paths.get()</code> erzeugt einen Pfad anhand der Parameter • <code>Stream<String> stream = Files.lines(path);</code> → Öffnet die Datei und gibt sie als Stream von <code>String</code> zurück → Bei Fehlern <code>IOException</code> aus <code>java.io</code> (importieren) • <code>String fileContentAsString = stream.reduce(String::concat);</code> → Erstellt aus allen Elementen des Streams einen einzelnen <code>String</code> 	8/95
	IntStreams	<ul style="list-style-type: none"> • Auch <code>LongStream</code> und <code>DoubleStream</code> • Handhabung analog zu normalen Streams • <code>IntStream stream1 = IntStream.of(1,2,3);</code> 	8/102
	Random Zahlen	<ul style="list-style-type: none"> • Klasse <code>Random</code> in <code>java.util</code> • <code>Random random = new Random();</code> (Erzeugung eines Objekts) • <code>random.nextDouble();</code> (oder <code>nextInt(...)</code>) • Bei <code>Double/Float</code> zwischen 0.0 und 0.1 • Bei <code>Int/Long</code> im Wertebereich • Bei Streams: <code>IntStream stream1 = new Random().ints();</code> → Füllt Stream mit <code>int</code> stream 	8/107

System Properties	Eigenschaften	<ul style="list-style-type: none"> • <code>java.lang.System</code> • Attribute der Umgebung, in der das Java-Programm läuft • Werden als <code>String</code> gespeichert, kriegen <code>String</code> übergeben 	8/84
	Methode	<ul style="list-style-type: none"> • <code>String homeDirectory = System.getProperty("user.home");</code> • Liefert <code>String</code> zurück, übernimmt <code>String</code> als Parameter 	8/86
	Mögliche Strings	<ul style="list-style-type: none"> • <code>"user.home"</code>: Name des Heimatverzeichnisses • <code>"user.dir"</code>: Arbeitsverzeichnis des Prozesses • <code>"user.name"</code>: Name des Nutzers • <code>"file.seperator"</code>: Zeichen zur Trennung von Pfadnamen • <code>"line.seperator"</code>: Zeichen zur Trennung von Zeilen 	8/87

Textdateien	Wissenswertes	<ul style="list-style-type: none"> • Textdatei besteht aus Zeichen (chars) • jeder Char ist zwei Byte groß • Suche nach bequemeren Zugriff für Textdateien (nicht byteweise) 	8/170
	Reader	<ul style="list-style-type: none"> • Klasse zum Einlesen von Textdateien • viele reader.read Methoden, Bsp. reader.read(char[] c); <ul style="list-style-type: none"> → Liest soviele Zeichen ein bis Array voll oder Datenquelle erschöpft → Gibt Anzahl der eingelesenen Chars aus • BufferedReader: reader.readLine(); <ul style="list-style-type: none"> → Richtet StringObjekt ein und liest ganze Zeile ein • Byteweises Einlesen in Zeichenweises Einlesen: <ul style="list-style-type: none"> → InputStream in = ...; → Reader reader = new InputStreamReader(in); 	8/172
	Writer	<ul style="list-style-type: none"> • Klasse zum Schreiben von Textdateien • writer.write(char); (schreibt einzelnen Char) • writer.write(String); (schreibt ganzen String) • Umwandlung von Byteweise in Zeichenweise analog zu Reader <ul style="list-style-type: none"> → OutputStreamWriter 	8/181

Threads	Wissenswertes	<ul style="list-style-type: none"> • Verweis auf Kapitel Nested Classes weiter oben • Organisation von parallel laufenden Prozessen • Beschleunigung eines Vorgangs aufgrund der Aufteilung in mehrere Prozesse • Aber nicht immer unbedingt schneller, Wissen notwendig • Starten und Vergessen (fire and forget") 	9/21
	Implementierung	<ul style="list-style-type: none"> • Inhalt eines Threads: <ul style="list-style-type: none"> → Klasse, die das Interface Runnable implementiert → Funktionale Methode public void run() {...} • Einrichtung der Runnable Klasse (9/29-44) • new Thread(runnable).start(); <ul style="list-style-type: none"> → Kein späterer Zugriff mehr nötig, Thread läuft ab sofort → Programmausführung besteht nun aus zwei Threads → Hier: Main Methode und gestarteter Thread 	9/23
	Methoden	<ul style="list-style-type: none"> • static currentThread <ul style="list-style-type: none"> → Keine Parameter, liefert ThreadObjekt zurück → Repräsentiert Thread, in dem die Methode aufgerufen wurde • dumpStack <ul style="list-style-type: none"> → Schreiben des Call-Stacks auf System.err • getAllStackTraces <ul style="list-style-type: none"> → Liefert Call-Stacks von allen aktiven Threads im Programm → Rückgabe als Map mit Threads als Keys • getId <ul style="list-style-type: none"> → Gibt ID vom Typ long zurück, bleibt gleich solange Thread aktiv • getName <ul style="list-style-type: none"> → Gibt Namen zurück • getPriority / setPriority <ul style="list-style-type: none"> → Prioritätswert vom Typ int • static sleep <ul style="list-style-type: none"> → Setzt Pause in Millisekunden 	9/51
	Streams	<ul style="list-style-type: none"> • Verknüpfung von Output in Input <ul style="list-style-type: none"> → Pipe = Verbindung zwischen Lesen und Schreiben → PipedOutputStream out = new PipedOutputStream(); → PipedInputStream in = new PipedInputStream(out); → Runnable runnable = new WriteRunner(out); → new Thread(runnable).start(); → Neuer Thread, der Output ausgibt, der in 2. Stream erzeugt wird → Hier müsste WriteRunner implements Runnable noch erstellt werden 	9/60
	Interferierende Threads	<ul style="list-style-type: none"> • Mehrere Threads Zugriff auf selbe Ressource <ul style="list-style-type: none"> → Reihenfolge unklar → z.B. printen beide auf Standard Output (System.out) 	9/79 → Vermischung komple

