Lernhinweise und Prüfungstipps

Die Lernfragen zu diesem Kapitel finden Sie unter:
https://narr.kwaest.io/s/1229

Was erwartet mich in diesem Kapitel?

In diesem Kapitel geht es um die Programmiersprache Java, die wichtigsten Begriffe, die Besonderheiten von Java, die üblichen Notationen und Konventionen in Java, die notwendigen Werkzeuge zur Softwareentwicklung sowie um das erste „Hallo Welt“-Programm in Java.

Welche Schlagwörter lerne ich kennen?

■ Java Software Development Kit (SDK), Java Development Kit (JDK) ■ Java Runtime Environment (JRE) ■ Plattformunabhängigkeit ■ Objektorientierung ■ Einfachheit ■ Netzwerkfähigkeit ■ Sicherheit ■ Offenheit ■ Java Standard Edition (JSE) ■ Java Enterprise Edition (JEE) ■ Java Embedded ■ Integrierte Entwicklungsumgebung (IDE) ■ Eclipse, NetBeans

Wofür benötige ich dieses Wissen?

Dieses Wissen wird benötigt, um (1) die zentralen Java-Begriffe zu verstehen, (2) Java als Programmiersprache (besser) einordnen zu können, (3) die Ausrichtungen der einzelnen Java-Editionen zu erkennen, und (4) die Vorteile der erforderlichen Werkzeuge, wie z.B. die integrierte Entwicklungsumgebung, zu sehen.

Welchen Prüfungstipp kann ich aus diesem Abschnitt ziehen?

In Prüfungen wird häufig gewünscht, die Besonderheiten von Java zu erklären und die einzelnen Java-Begriffe oder -Editionen zu erläutern.

1.1 Historie

Java wurde 1995 von einem Team von Entwicklern um James Gosling und Billy Joy im Auftrag der Firma SUN Microsystems entworfen und umgesetzt. Im Jahr 2010 wurde SUN Microsystems von Oracle übernommen. Im Zug dieser Transaktion wurden auch alle intellektuellen Rechte an Java, z.B. die Patente, Lizenzen und Urheberrechte, an Oracle übertragen.

Die Entwicklung von Java wurde stark durch andere Programmiersprachen, wie z.B. C++ und Smalltalk, beeinflusst. Java selbst hat wiederum zu einer Vielzahl von neuen Programmiersprachen, wie z.B. Groovy, AspectJ, Clojure oder Scala, hervorgebracht, die die zugrundeliegende Architektur, z.B. die virtuelle Java-Maschine, übernehmen oder einzelne Aspekte vertiefen.

Die Weiterentwicklung von Java erfolgt im Rahmen des Java Community Processes (JCP), an dem sich viele renommierte IT-Hersteller, IT-Dienstleister und Softwareunternehmen beteiligen. Trotz der Einbindung anderer Unternehmen bleibt Oracle jedoch die bestimmende Kraft bei der Weiterentwicklung von Java. Vorschläge zu einer neuen Version einer bereits existierenden oder einer innovativen, erstmaligen Technologie werden in Form Java Specification Requests (JSR) definiert, der den jeweils aktuellen Stand der Diskussion definiert (einsehbar auf www.jcp.org). Die daraus resultierenden technischen Spezifikationen der verschiedenen Java-Technologien können auch auf den einschlägigen Webseiten der Firma Oracle eingesehen werden.

1.2 Begriffe

Das Java Software Development Kit (Java SDK oder auch manchmal als JDK bezeichnet) ist die technische Voraussetzung, um in Java programmieren zu können. Es ist über die Webseiten von Oracle für verschiedene Hardware-Plattformen (z.B. Windows, Linux, MacOS) kostenlos verfügbar, es existieren darüber hinaus aber auch noch weitere Implementierungen von anderen Herstellern.

Das **Java SDK** umfasst zum einen den Compiler, der den Java-Quellcode in einen plattformübergreifenden Zwischencode (Bytecode) übersetzt, und zum anderen die **Java-**Laufzeitumgebung (**Java Runtime Environment**, **JRE**) inklusive der zahlreichen Klassenbibliotheken und dem Interpreter, der den Zwischencode in die jeweilige Zielplattform überführt. **Java Development Kit (JDK)** wird oft synonym zu Java SDK verwendet.

Aus diesem Ansatz ergeben sich zwei wesentliche Vorteile:

- erstens, ein Anwender kann Zwischencode problemlos von einer Hardware-Plattform auf eine andere übertragen, wo er dort dann interpretiert wird, und
- zweitens, ein Hersteller muss für eine neue Hardware-Plattform nur eine entsprechende Java-Laufzeitumgebung bereitstellen, was weit weniger aufwändig ist, als das gesamte Java SDK zu portieren.

Der Quelltext eines Java-Programms könnte prinzipiell zwar mit Hilfe eines Texteditors erstellen werden, aber dies wäre nur wenig sinnvoll, da weitere unterstützende Funktionen, wie z.B. die Ausführung oder das Debugging des Java-Codes, über die Kommandozeile aufgerufen werden müssten. Aus diesem Grund gibt es integrierte Entwicklungsumgebungen („Integrated Development Environments", IDE), wie z.B. Eclipse oder NetBeans, die all diese Funktionalitäten in einer gemeinsamen Benutzeroberfläche bündeln und als kostenlose Versionen verfügbar sind.

1.3 Besonderheiten von Java

Die wesentlichen Differenzierungsfaktoren von Java zum Zeitpunkt seines Entwurfs waren:

- Plattformunabhängigkeit
- Objektorientierung
- Einfachheit
- Netzwerkfähigkeit
- Sicherheit
- Offenheit/Open Source

Plattformunabhängigkeit bedeutet, dass der Java-Quellcode in den Zwischencode übersetzt wird und dann von jeder Java-Laufzeitumgebung, unabhängig von der Betriebssystem-Plattform, ausgeführt werden kann.

Die Java-Laufzeitumgebung ist (immer) eine Voraussetzung für die Ausführung von Java-Programmen und stellt eine Abstraktion von der konkreten Hardware dar. Insbesondere die plattformunabhängige Verfügbarkeit einer grafischen Benutzeroberfläche war ein wichtiges Alleinstellungsmerkmal bei der Markteinführung von Java. Grafische Benutzeroberflächen mit Java werden in Schritt 13 eingeführt.

Objektorientierung basiert darauf, dass Klassen definiert werden, von denen Instanzen oder Objekte erzeugt werden. Das Programm besteht dann in dem Versenden von Nachrichten zwischen den einzelnen Objekten (bzw. dem Aufruf von Methoden laut der Java-Terminologie).

Das Prinzip der Objektorientierung war zum damaligen Zeitpunkt bereits in einigen Programmiersprachen, z.B. Smalltalk, C++ oder Objective-C umgesetzt, aber erst durch Java erlangte die konsequente und durchgängige objektorientierte Programmierung die nötige Verbreitung und Unterstützung.

Typische Mechanismen von objektorientierten Programmiersprachen sind Klassen, die Vererbung und der Polymorphismus.

In Java wird das objektorientierte Paradigma ein wenig abgemildert, wenn es um die einfachen Datentypen geht, die nicht als Klassen implementiert sind. Die Konzepte der objektorientierten Programmierung werden in Schritt 7 betrachtet.

Unter **Einfachheit** versteht man bei einer Programmiersprache, dass die zugrundeliegenden Konstrukte übersichtlich, verständlich und leicht zu erlernen sind, so dass bereits mit wenigen Schlüsselwörtern und -konzepten eine große Ausdrucksfähigkeit erreicht wird.

Die Einfachheit von Java kommt in zwei Aspekten zum Tragen: zum einen orientiert sich die Syntax von Java an C und zum zweiten besteht Java aus einem „Sprachkern“ und Klassenbibliotheken. Das Besondere ist hierbei, dass der Sprachkern nur wenige, grundlegende Sprachkonstrukte und Schlüsselwörter enthält, so dass er leicht erlernbar und verständlich ist. Die Auslagerung vieler Funktionalitäten in die Klassenbibliotheken erlaubt eine große Flexibilität und kommt auch dadurch zum Ausdruck, dass es drei Versionen (Standard, Enterprise und Embedded) gibt, die sich in erster Linie aufgrund der Klassenbibliotheken unterscheiden. Der Sprachkern von Java wird in den Schritten 2 bis 4 vorgestellt.

Eine Programmiersprache gilt als **netzwerkfähig**, wenn die grundlegenden Funktionalitäten zum Aufbau und zur Beendigung einer Netzwerkverbindung integriert sind, z.B. über eine entsprechende Standard-Klassenbibliothek.

Die Netzwerkfähigkeit war von Anfang an in Java integriert und motivierte deshalb den Einsatz von Java in verteilten Anwendungen und bei der sich damals erst entwickelnden Web-Programmierung. Aufbauend auf diesen grundlegen-

den Features entwickelte sich sehr schnell die Java Enterprise Edition (JEE), die insbesondere für verteilte Systeme und große, geschäftskritische Anwendungen in Unternehmen ausgelegt ist.

Aufgrund der einfachen Erweiterbarkeit von Java kamen in den vergangenen Jahren immer mehr Klassen und Funktionen zur Netzwerkprogrammierung hinzu, die einfach in die bisherigen Klassenbibliotheken integriert oder in separate ausgelagert werden konnten.

Die **Sicherheit** einer Programmiersprache hängt von unterschiedlichen Aspekten ab: von einer zuverlässigen Speicherverwaltung, über die Typprüfung aller Variablen bis hin zu einer konsequenten Fehlerbehandlung, der Validierung des erzeugten Codes oder die eventuellen Einschränkungen bei der Ausführung der Java-Anwendung.

Zahlreiche Sicherheitsmechanismen sind von vorneherein in Java eingebunden. Das automatische Speichermanagement erlaubt es, nicht mehr benötigte Speicherbereiche nach Gebrauch wieder frei zu geben und so „memory leaks“ zu vermeiden. Memory leaks sind Speicherbereiche, die für die Nutzung eines Programms reserviert sind, die aber anschließend nicht mehr frei gegeben werden.

Alle Variablen müssen deklariert, typisiert und initialisiert werden, bevor sie verwendet werden dürfen. Dies reduziert deutlich die Gefahr von falschen oder unvollständigen Werten.

Die integrierte Fehlerbehandlung dank der Exceptions erlaubt es, auch auf Störungen des Programmablaufs zur Laufzeit entsprechend zu reagieren und diese sogar ggf. zu beheben. Die Ausnahmebehandlung wird in Schritt 11 diskutiert.

Die Sicherheit beschränkt sich dabei nicht nur auf die Programmierung, sondern sie kann sich insbesondere auf die Ausführung des Java-Codes erstrecken. Zur Abwehr von Schadsoftware kann überprüft werden, ob der Code selbst verändert wurde, oder es können sogar Sicherheitsregeln definiert werden, die die Ausführungs- und Zugriffsrechte der Anwendung einschränken, um so zu verhindern, dass ein potentieller Angreifer weitergehende Rechte erwirbt.

Als **Offenheit** wird die Veröffentlichung aller relevanten Schnittstellen, Formate und Technologien bezeichnet.

Unter **Open Source** versteht man in der Regel, dass der Quellcode öffentlich verfügbar gemacht wird, unter einer entsprechenden offenen Lizenz steht, also z.B. verändert, kopiert und weitergegeben werden kann, und sich eine Community, eventuell in Partnerschaft mit kommerziellen Firmen, um die Weiterentwicklung kümmert.