Kursbeschreibung (CX-310-065)

Sun Certified Programmer for the Java Platform, Standard Edition 6 (CX-310-065)

Preis exkl. MWSt (EUR)	
235,00	Zertifizierung bestellen

Übersicht
Sun Certified Programmer for the Java Platform, Standard Edition 6 Die Zertifizierungsprüfung zum Sun Certified Programmer for Java Platform, Standard Edition 6, richtet sich an Programmierer mit Erfahrung in der Programmiersprache Java. Mit dieser Zertifizierung weist der Programmierer seine Kenntnisse der grundlegenden Syntax und Struktur der Programmiersprache Java nach und belegt die Fähigkeit, Java-Anwendungen zu erstellen, die auf Server- und Desktop-Systemen unter Java SE 6 ausgeführt werden.

Details

- Delivered at: Ort: Autorisierte Prometric-Testzentren weltweit
- Prerequisites: Keine
- Other exams/assignments required for this certification: Für diese Zertifizierung vorausgesetzte andere Prüfungen/Aufgaben: Keine
- Exam type: Prüfungstyp: Multiple-Choice-Fragen, Drag-and-Drop-Fragen
- Number of questions: 72
- Pass score: 65 % (47 von 72 Fragen)
- Time limit: 210 Minuten

Languages
Englisch

Voraussetzungen
Keine

Prüfungsziele

Teil 1: Deklarationen, Initialisierung und Scoping

- Entwickeln von Code, der Klassen (einschließlich abstrakter und aller Formen geschachtelter Klassen), Schnittstellen und Enums deklariert und eine korrekte Verwendung von Package- und Importanweisungen (einschließlich statischer Importe) enthält.
- Entwickeln von Code, der eine Schnittstelle deklariert. Entwickeln von Code, der eine oder mehrere Schnittstellen implementiert oder erweitert. Entwickeln von Code, der eine abstrakte Klasse deklariert. Entwickeln von Code, der eine abstrakte Klasse erweitert.
- Entwickeln von Code, der Primitive, Arrays, Enums und Objekte als statische, Instanz- und lokale Variablen deklariert, initialisiert und verwendet. Dazu gehört auch die Verwendung zulässiger Identifikatoren für Variablennamen.
- Entwickeln von Code, der statische und nicht-statische Methoden deklariert, und gegebenenfalls Verwenden von Methodennamen, die den JavaBeans-Benennungsstandards entsprechen. Darüber hinaus Entwickeln von Code, der eine Liste von Argumenten variabler Länge deklariert und verwendet.
- Anhand eines Codebeispiels Bestimmen, ob eine Methode das Overriding bzw. Overloading einer anderen Methode korrekt ausführt, und Identifizieren der zulässigen Return-Werte (einschließlich kovarianter Return-Werte) für diese Methode.
- Am Beispiel eines vorgegebenen Sets von Klassen und Superklassen Entwickeln von Konstruktoren für eine oder mehrere dieser Klassen. Am Beispiel einer vorgegebenen Klassendeklaration feststellen, ob ein Standardkonstruktor erstellt wird und, wenn ja, das Verhalten dieses Konstruktors bestimmen. Am Beispiel einer Liste geschachtelter oder nicht-geschachtelter Klassen Schreiben von Code zum Instantiieren dieser Klasse.

Teil 2: Ablaufsteuerung

- Entwickeln von Code, der eine if- oder switch-Anweisung implementiert, und Erkennen der zulässigen Argumenttypen für diese Anweisungen.
- Entwickeln von Code, der alle Formen von Schleifen und Iteratoren implementiert, einschließlich for- und erweiterter for-Schleifen (for-each) sowie do, while, labels, break und continue. Erläutern der Werte, die die

Schleifenzählervariablen vor und nach der Ausführung der Schleife annehmen.
- Entwickeln von Code, der Assertions ordnungsgemäß nutzt, sowie Unterscheiden zwischen passender und unpassender Verwendung von Assertions.
- Entwickeln von Code mit korrekter Nutzung von Ausnahmen und Ausnahmebehandlungsklauseln (try, catch, finally) sowie Deklaration von Methoden und Override-Methoden, die Ausnahmen auslösen.
- Erkennen, welche Wirkung eine Ausnahme hat, die an einer spezifizierten Stelle in einem Codefragment auftritt. Hinweis: Bei der Ausnahme kann es sich um eine Runtime-Ausnahme, eine Checked Exception oder einen Fehler handeln.
- Erkennen von Situationen, die eine der folgenden Erscheinungen auslösen: ArrayIndexOutOfBoundsException, ClassCastException, IllegalArgumentException, IllegalStateException, NullPointerException, NumberFormatException, AssertionError, ExceptionInInitializerError, StackOverflowError oder NoClassDefFoundError. Erkennen, welche davon durch die Virtual Machine ausgelöst werden und in welchen Situationen andere durch das Programm ausgelöst werden sollten.

Teil 3: API-Inhalte

- Entwickeln von Code, der die primitiven Wrapper-Klassen nutzt (Boolean, Character, Double, Integer usw.), und/oder Autoboxing und Unboxing. Darlegen der Unterschiede zwischen den Klassen String, StringBuilder und StringBuffer.
- Am Beispiel eines vorgegebenen Szenarios zur Navigation in Dateisystemen sowie zum Lesen aus Dateien, Schreiben in Dateien oder Interagieren mit dem Benutzer eine korrekte Lösung unter Verwendung der folgenden Klassen aus java.io (auch in Kombination) entwickeln: BufferedReader, BufferedWriter, File, FileReader, FileWriter, PrintWriter und Console.
- Entwickeln von Code, der Objekte mithilfe der folgenden APIs aus java.io serialisiert und/oder deserialisiert: DataInputStream, DataOutputStream, FileInputStream, FileOutputStream, ObjectInputStream, ObjectOutputStream und Serializable.
- Verwenden der Standard-J2SE-APIs im Package java.text zum korrekten Formatieren oder Parsen von Daten, Zahlen und Währungswerten für eine bestimmte Sprachumgebung. Am Beispiel eines vorgegebenen Szenarios Bestimmen der geeigneten Methoden für die Verwendung der Standardsprachumgebung oder einer spezifischen Sprachumgebung. Beschreiben von Sinn und Verwendungszweck der Klasse java.util.Locale.
- Schreiben von Code, der zum Formatieren oder Parsen von Strings oder Streams Standard-J2SE-APIs in den Packages java.util und java.util.regex verwendet. Schreiben von Code für Strings, der die Klassen Pattern und Matcher sowie die Methode String.split verwendet. Erkennen und Nutzen regulärer Ausdrucksmuster für Abgleichsoperationen (beschränkt auf: . (Punkt), * (Stern), + (Plus), ?, \d, \s, \w, [], ()). Die Verwendung von *, +, und ? bleibt auf Greedy Quantifiers beschränkt und der Operator „runde Klammern“ wird nur für die Gruppierung eingesetzt, nicht zum Erfassen von Inhalten beim Abgleich. Schreiben von Code für Streams, der die Klassen Formatter und Scanner sowie die Methoden PrintWriter.format/printf verwendet. Erkennen und Verwenden von Formatierungsparametern (beschränkt auf: %b, %c, %d, %f, %s) in Formatierungs-Strings.

Teil 4: Concurrency (Nebenläufigkeit)

- Schreiben von Code zum Definieren, Instantiieren und Starten neuer Threads unter Verwendung von java.lang.Thread und java.lang.Runnable.
- Erkennen der Statusformen, die ein Thread annehmen kann, und Identifizieren von Möglichkeiten, wie ein Thread von einem Status in einen anderen übergehen kann.
- Schreiben von Code am Beispiel eines vorgegebenen Szenarios, in dem zum Schutz von statischen oder Instanzvariablen vor Problemen im Zusammenhang mit gleichzeitigen Zugriffen Objektsperren korrekt eingesetzt werden.
- Schreiben von Code, in dem wait, notify oder notifyAll korrekt eingesetzt werden, und zwar am Beispiel eines vorgegebenen Szenarios.

Teil 5: OO-Konzepte

- Entwickeln von Code, der enge Kapselung, lockere Kopplung und hohe Kohäsion in Klassen implementiert und den Nutzen dieser Merkmale beschreibt.
- Entwickeln von Code, der die Verwendung von Polymorphismus demonstriert, und zwar am Beispiel eines vorgegebenen Szenarios. Feststellen, wann Casting erforderlich ist, und Erkennen von Compiler- im Gegensatz zu Runtime-Fehlern, die im Zusammenhang mit dem Objektreferenz-Casting auftreten.
- Erläutern der Wirkung von Modifikatoren auf die Vererbung, und zwar im Hinblick auf Konstruktoren, Instanz- oder statische Variablen und Instanz- oder statische Methoden.
- Entwickeln von Code, der Overloading-Methoden deklariert und/oder aufruft, sowie von Code, der Superklassen- oder Overloading-Konstruktoren deklariert und/oder aufruft, und zwar am Beispiel eines vorgegebenen Szenarios.
- Entwickeln von Code, der „ist-ein“- und/oder „hat-ein“-Beziehungen implementiert.

Teil 6: Collections/Allgemeines

- Am Beispiel eines vorgegebenen Designszenarios Entscheiden, welche Collections-Klassen und/oder Schnittstellen zur korrekten Implementierung dieses Designs eingesetzt werden sollten (einschließlich Verwendung der Comparable-Schnittstelle).
- Unterscheiden zwischen richtigen und falschen Overrides der entsprechenden Methoden hashCode und equals sowie Erläutern des Unterschieds zwischen == und der Methode equals.
- Schreiben von Code, der die generischen Versionen der Collections-API nutzt, insbesondere die Set-, List- und Map-Schnittstelle und die Implementierungsklassen. Erkennen der Beschränkungen der nicht-generischen Collections-API und Erläutern, wie man Code so umschreiben („Refactor“) kann, dass er die generischen Versionen verwendet. Schreiben von Code, der die Schnittstellen NavigableSet und NavigableMap verwendet.
- Entwickeln von Code, der Typparameter in Klassen-/Schnittstellendeklarationen, Instanzvariablen, Methodenargumenten und Return-Typen sachgemäß nutzt. Schreiben von generischen Methoden oder von Methoden, die Platzhaltertypen nutzen. Erkennen der Ähnlichkeiten und Unterschiede zwischen diesen beiden Strategien.
- Schreiben von Code zur Listenmanipulation unter Verwendung der Funktionen im Package java.util: Sortieren, Ausführen einer binären Suche oder Konvertieren der Liste in ein Array. Schreiben von Code zur Array-Manipulation unter Verwendung der Funktionen im Package java.util: Sortieren, Ausführen einer binären Suche oder Konvertieren des Array in eine Liste. Beeinflussen der Listen- und Array-Sortierung mithilfe der Schnittstelle java.util.Comparator und java.lang.Comparable. Darüber hinaus Erkennen der Wirkung einer „natürlichen Ordnung“ („Natural Ordering“) primitiver Wrapper-Klassen und des java.lang.String auf die Sortierung.

Teil 7: Grundlagen

- Schreiben von Code, der mithilfe der geeigneten Zugriffsmodifikatoren, Package-Deklarationen und Importanweisungen mit dem Beispielcode interagiert (per Zugriff oder Vererbung), und zwar am Beispiel eines vorgegebenen Codes und eines Szenarios.
- Am Beispiel einer vorgegebenen Klasse und Befehlszeile Erkennen des zu erwartenden Runtime-Verhaltens.
- Erkennen, welche Wirkung die Übergabe an Methoden, die Zuweisungen oder andere modifizierende Operationen auf die Parameter ausführen, auf Objektreferenzen und primitive Werte hat.
- Erkennen der Stelle in einem Codebeispiel, ab der ein Objekt vom Bereinigungsmechanismus erfasst werden kann. Feststellen, welches Verhalten vom Bereinigungssystem garantiert bzw. nicht garantiert wird, und Erkennen der Verhalten der Object.finalize()-Methode.
- Am Beispiel eines vollständig qualifizierten Namens einer Klasse, die innerhalb und/oder außerhalb einer JAR-Datei bereitgestellt wird, Konstruieren der geeigneten Verzeichnisstruktur für diese Klasse. Anhand eines Codebeispiels und eines Klassenpfads Feststellen, ob der Klassenpfad eine erfolgreiche Kompilierung des Codes zulässt.
- Schreiben von Code, in dem die passenden Operatoren korrekt angewendet werden, darunter Zuweisungsoperatoren (beschränkt auf: =, +=, -=), arithmetische Operatoren (beschränkt auf: +, -, *, /, %, ++, --), relationale Operatoren (beschränkt auf: <, <=, >, >=, ==, !=), den Operator instanceof, logische Operatoren (beschränkt auf: &, |, ^, !, &&, ||) und den konditionalen Operator (? :), um auf diese Weise das angestrebte Ergebnis zu erzielen. Schreiben von Code, der die Gleichwertigkeit von zwei Objekten oder zwei Primitiven bestimmt.