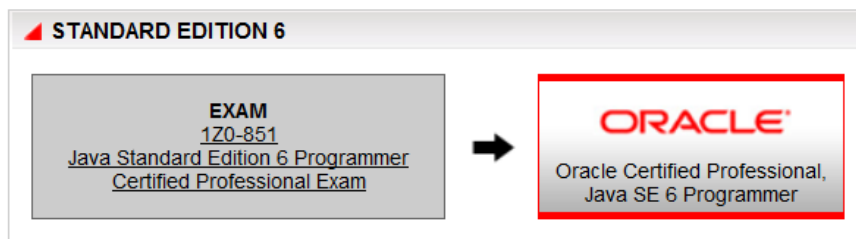


JAVA SE 6 PROGRAMMER J2SE 6.0

Oracle Certified Professional Java Programmer

Esta certificación está dirigida a programadores con experiencia en el lenguaje de programación Java. La obtención de este título certifica que el programador comprende la sintaxis y la estructura básica del lenguaje de programación Java y puede crear aplicaciones Java que se ejecutan en sistemas de servidor y escritorio utilizando J2SE 6.0.

Para alcanzar esta certificación, los candidatos deben superar el examen 1Z0-851 en un Test Center de VUE. Cenec Málaga es Test Center de VUE. No es necesario tener una certificación de nivel Associate en Plataforma Java antes de hacer el examen de esta certificación. (Anteriormente Sun Certified Java Programmer (SCJP))



Contenidos:

Sección 1: Declaraciones, inicialización y ámbito

- Desarrollar código que declare clases (incluidas clases abstractas y todas las formas de clases anidadas), interfaces y enums, y que incluya el uso apropiado de instrucciones package e import (incluidas importaciones estáticas).
- Desarrollar código que declare una interfaz. Desarrollar código que implemente o amplíe una o más interfaces. Desarrollar código que declare una clase abstracta. Desarrollar código que amplíe una clase abstracta.
- Desarrollar código que declare, inicialice y use primitivas, arreglos, enums y objetos como variables estáticas, de instancia y locales. Asimismo, utilizar identificadores válidos para los nombres de variable.
- Desarrollar código que declare métodos tanto estáticos como no estáticos y, si resulta adecuado, que utilice nombres de método con nomenclatura de JavaBeans. Además, desarrollar código que declare y utilice una lista de argumentos de longitud variable.
- Con un ejemplo de código dado, determinar si un método está sobrescribiendo o sobrecargando correctamente otro método e identificar valores de retorno válidos (incluidos valores de retorno covariantes) para el método.
- A partir de una serie de clases y superclases, desarrollar constructores para una o varias clases. Con una declaración de clase dada, determinar si se creará un constructor predeterminado

y, en caso afirmativo, determinar su comportamiento. Con una lista de clases anidadas y no anidadas, escribir código para crear instancias de la clase.

Sección 2: Control de flujo

- Desarrollar código que implemente una instrucción if o switch e identificar tipos de argumentos válidos para estas instrucciones.
- Desarrollar código que implemente todas las formas de ciclos e iteradores, incluidos el uso de for, el ciclo mejorado (for-each), do, while, labels, break y continue. Indicar los valores que adoptan las variables de control del ciclo durante y después de la ejecución del ciclo.
- Escribir código que utilice afirmaciones y distinga entre el uso apropiado e inapropiado de las afirmaciones.
- Desarrollar código que utilice excepciones y cláusulas de manejo de excepciones (try, catch, finally), y declarar métodos y sobrescribir métodos que generen excepciones.
- Reconocer el efecto que produce una excepción que se genera en un punto dado de un fragmento de código. Puede ser una excepción runtime, una excepción comprobada (checked) o un error.
- Reconocer las situaciones en las que se generará alguna de las siguientes excepciones: `ArrayIndexOutOfBoundsException`, `ClassCastException`,

IllegalArgumentException, IllegalStateException, NullPointerException, NumberFormatException, AssertionError, ExceptionInInitializerError, StackOverflowError o NoClassDefFoundError. Saber cuáles genera la máquina virtual y en qué situaciones deberían generarse programáticamente otras excepciones.

Sección 3: Contenido del API

- Desarrollar código que utilice las clases wrapper primitivas (como booleano, carácter, doble, entero, etc.) y/o autoboxing y unboxing. Describir las diferencias entre las clases String, StringBuilder y StringBuffer.
- En una situación en la que se requiera desplazarse por los file systems, leer archivos o escribir en archivos, desarrollar la solución adecuada mediante el uso de las siguientes clases (o una combinación de ellas) del paquete java.io: BufferedReader, BufferedWriter, File, FileReader, FileWriter y PrintWriter.
- Desarrollar código que serialice y/o deserialice objetos mediante el uso de las siguientes API de java.io: DataInputStream, DataOutputStream, FileInputStream, FileOutputStream, ObjectInputStream, ObjectOutputStream y Serializable.
- Utilizar API de J2SE estándar del paquete java.text para asignar formato y analizar correctamente fechas, números y valores de moneda de una configuración regional específica. En una situación dada, determinar los métodos que se deben utilizar si se desea emplear la configuración regional predeterminada u otra específica. Describir el objetivo y la utilidad de la clase java.util.Locale.
- Escribir código que utilice API de J2SE estándar de los paquetes java.util y java.util.regex para asignar formato o analizar cadenas o secuencias. Para cadenas, escribir código que utilice las clases Pattern y Matcher y el método String.split. Reconocer y utilizar patrones de expresión regulares para establecer la coincidencia (limitado a: . (punto), * (asterisco), + (signo más), ?, \d, \s, \w, [], ()). El uso de *, + y ? se limitará a cuantificadores greedy y el operador de paréntesis sólo se empleará como mecanismo de agrupación, en lugar de para capturar contenido durante la coincidencia. Para secuencias, escribir código utilizando las clases Formatter y Scanner y los métodos PrintWriter.format/printf. Reconocer y utilizar los parámetros de formato (limitado a: %b, %c, %d, %f, %s) en cadenas de formato.

Sección 4: Concurrencia

- Escribir código para definir, instanciar e iniciar nuevos threads utilizando java.lang.Thread y java.lang.Runnable.
- Reconocer los estados en los que puede existir un thread, e identificar condiciones en las que puede pasar de un estado a otro.
- En una situación dada, escribir código que utilice correctamente el bloqueo de objetos para proteger las variables estáticas o de instancia de los problemas de acceso concurrente.
- En una situación dada, escribir código que utilice correctamente wait, notify o notifyAll.

Sección 5: Conceptos de la programación OO

- Desarrollar código que implemente encapsulación estricta, emparejamiento ligero y gran cohesión en las clases, y describir las ventajas que ofrece.
- En una situación dada, desarrollar código que demuestre el uso del polimorfismo. También determinar cuándo se va a necesitar la conversión de tipos y distinguir los errores del compilador de los errores de tiempo de ejecución relacionados con la conversión de referencias de objeto.
- Explicar el efecto de los modificadores en la herencia en lo que se refiere a constructores, variables de instancia o estáticas y métodos de instancia o estáticos.

- En una situación dada, desarrollar código que declare y/o llame a métodos sobrescritos o sobrecargados y código que declare y/o llame a constructores de superclase, sobrescritos o sobrecargados.
- Desarrollar código que implemente relaciones "is-a" y/o "has-a".

Sección 6: Colecciones / Genéricos

- En una situación de diseño dada, determinar las clases o interfaces de colección que deberían utilizarse para implementar ese diseño de forma adecuada, incluido el uso de la interfaz Comparable.
- Distinguir entre los valores de reemplazo correctos e incorrectos de los métodos hashCode y equals correspondientes, y explicar la diferencia entre == y el método equals.
- Escribir código que utilice las versiones genéricas de las colecciones del API, en particular las interfaces Set, List y Map y las clases de implementación. Identificar las limitaciones de las colecciones API no genéricas y cómo refactorizar código para utilizar versiones genéricas. Escribir código que use las interfaces NavigableSet y NavigableMap.
- Desarrollar código que utilice correctamente los parámetros de tipo en las declaraciones de clase/interfaz, las variables de instancia, los argumentos de métodos y los tipos de retorno. Escribir métodos genéricos o métodos que utilicen comodines y comprender las similitudes y diferencias entre estos dos métodos.
- Utilizar las funciones del paquete java.util para escribir código destinado a manipular una lista para ordenarla, realizando un búsqueda binaria o convirtiéndola en un arreglo. Utilizar las funciones del paquete java.util para escribir código destinado a manipular un arreglo ordenándolo, realizando un búsqueda binaria o convirtiéndolo en una lista. Utilizar las interfaces java.util.Comparator y java.lang.Comparable para afectar la clasificación de las listas y los arreglos. Además, conocer el efecto que produce la "ordenación natural" de las clases primitivas wrapper y java.lang.String al ordenar.

Sección 7: Aspectos básicos

- Dado un ejemplo de código y una situación, escribir código que utilice los modificadores de acceso, las declaraciones package y las instrucciones import adecuadas para interactuar (a través de acceso o herencia) con el código del ejemplo.
- Con un ejemplo de una clase y una línea de comandos, determinar el comportamiento previsto del tiempo de ejecución.
- Determinar cómo afectan las referencias de objeto y valores primitivos cuando son pasados a métodos que realizan asignaciones u otras modificaciones en los parámetros.
- Con un ejemplo de código reconocer el momento en el que un objeto se convierte en elegible del garbage collection, determinar que está y que no está garantizado por el sistema del garbage collection y reconocer los comportamientos del método Object.finalize().
- Con el nombre completo de una clase que se implementa dentro y/o fuera de un archivo JAR, construir la estructura de directorios adecuada a dicha clase. Con un ejemplo de código y una ruta de clase dados, determinar si la ruta de clase permite que el código se compile correctamente.
- Escribir código que aplique correctamente los operadores adecuados para obtener el resultado deseado, incluidos los operadores de asignación (limitados a: =, +=, -=), los operadores aritméticos (limitado a: +, -, *, /, %, ++, --), los operadores relacionales (limitado a: <, <=, >, >=, ==, !=), el operador instanceof, los operadores lógicos (limitado a: &, |, ^, !, &&, ||) y el operador condicional (? :), para producir un resultado deseado. Escribir código que determine la igualdad de dos objetos o dos primitivas.