Certification Objectives

Sun Certified Programmer for the Java 2 Platform, Standard Edition 5.0

Section 1: Declarations, Initialization and Scoping

1.1. Develop code that declares classes (including abstract and all forms of nested classes), interfaces, and enums, and includes the appropriate use of package and import statements (including static imports).

1.2. Develop code that declares an interface. Develop code that implements or extends one or more interfaces. Develop code that extends an abstract class.

1.3. Develop code that declares, initializes, and uses primitives, arrays, enums, and objects as static, instance, and local variables. Also, use legal identifiers for variable names.

1.4. Develop code that declares both static and non-static methods, and - if appropriate - use method names that adhere to the JavaBeans naming standards. Also develop code that declares and uses a variable-length argument list.

1.5. Given a code example, determine if a method is correctly overriding or overloading another method, and identify legal return values (including covariant returns), for the method.

1.6. Given a set of classes and superclasses, develop constructors for one or more of the classes. Given a class declaration, determine if a default constructor will be created, and if so, determine the behavior of that constructor. Given a nested or non-nested class listing, write code to instantiate the class.

Section 2: Flow Control

2.1. Develop code that implements an if or switch statement; and identify legal argument types for these statements.

2.2. Develop code that implements all forms of loops and iterators, including the use of for, the enhanced for loop (for-each), do, while, labels, break, and continue; and explain the values taken by loop counter variables during and after loop execution.

2.3. Develop code that makes use of assertions, and distinguish appropriate from inappropriate uses of assertions.

2.4. Develop code that makes use of exceptions and exception handling clauses (try, catch, finally), and declares methods and overriding methods that throw exceptions.

2.5. Recognize the effect of an exception arising at a specified point in a code fragment. Note that the exception may be a runtime exception, a checked exception, or an error.

2.6. Recognize situations that will result in any of the following being thrown: ArrayIndexOutOfBoundsException, ClassCastException, IllegalArgumentException, IllegalStateException, NullPointerException, NumberFormatException, AssertionError, ExceptionInInitializerError, StackOverflowError or NoClassDefFoundError.?Understand which of these are thrown by the virtual machine and recognize situations in which others should be thrown programatically.

Section 3: API Contents

3.1. Develop code that uses the primitive wrapper classes (such as Boolean, Character, Double, Integer, etc.), and/or autoboxing & unboxing. Discuss the differences between the String, StringBuilder, and StringBuffer classes.

3.2. Given a scenario involving navigating file systems, reading from files, or writing to files, develop the correct solution using the following classes (sometimes in combination), from java.io: BufferedReader,BufferedWriter, File, FileReader, FileWriter and PrintWriter.

3.3. Develop code that serializes and/or de-serializes objects using the following APIs from java.io: DataInputStream, DataOutputStream, FileInputStream, FileOutputStream, ObjectInputStream, ObjectOutputStream and Serializable. Additionally, develop Serializable classes that correctly declare and use transient variables and private readObject and writeObject methods. Given a scenario and/or code example, recognize when, if, and which constructors will be called in an object's inheritance chain during deserialization.

3.4. Use standard J2SE APIs in the java.text package to correctly format or parse dates, numbers, and currency values for a specific locale; and, given a scenario, determine the appropriate methods to use if you want to use the default locale or a specific locale. Describe the purpose and use of the java.util.Locale class.

3.5. Write code that uses standard J2SE APIs in the java.util and java.util.regex packages to format or parse strings or streams. For strings, write code that uses the Pattern and Matcher classes and the String.split method. Recognize and use regular expression patterns for matching (limited to: . (dot), * (star), + (plus), ?, \d, \s, \w, [], ()). The use of *, +, and ? will be limited to greedy quantifiers, and the parenthesis operator will only be used as a grouping mechanism, not for capturing content during matching. For streams, write code using the Formatter and Scanner classes and the PrintWriter.format/printf methods. Recognize and use formatting parameters (limited to: %b, %c, %d, %f, %s) in format strings.

Section 4: Concurrency

4.1. Write code to define, instantiate, and start new threads using both java.lang.Thread and java.lang.Runnable.

4.2. Recognize the states in which a thread can exist, and identify ways in which a thread can transition from one state to another.

4.3. Given a scenario, write code that makes appropriate use of object locking to protect static or instance variables from concurrent access problems.

4.4. Given a scenario, write code that makes appropriate use of wait, notify, or notifyAll.

Section 5: OO Concepts

5.1. Develop code that implements tight encapsulation, loose coupling, and high cohesion in classes, and describe the benefits.

5.2. Given a scenario, develop code that demonstrates the use of polymorphism. Further, determine when casting will be necessary and recognize compiler vs. runtime errors related to object reference casting.

5.3. Explain the effect of modifiers on inheritance with respect to constructors, instance or static variables, and instance or static methods.

5.4. Given a scenario, develop code that declares and/or invokes overridden or overloaded methods and code that declares and/or invokes superclass, overridden, or overloaded constructors.

5.5. Develop code that implements "is-a" and/or "has-a" relationships.

Section 6: Collections / Generics

6.1. Given a design scenario, determine which collection classes and/or interfaces should be used to properly implement that design, including the use of the Comparable interface.

6.2. Distinguish between correct and incorrect overrides of corresponding hashCode and equals methods, and explain the difference between == and the equals method.

6.3. Write code that uses the generic versions of the Collections API, in particular, the Set, List, Queue and Map interfaces and implementation classes. Recognize the limitations of the non-generic Collections API and how to refactor code to use the generic versions.

6.4. Develop code that makes proper use of type parameters in class/interface declarations, instance variables, method arguments, and return types; and write generic methods or methods that make use of wildcard types and understand the similarities and differences between these two approaches.

6.5. Use capabilities in the java.util package to write code to manipulate a list by sorting, performing a binary search, or converting the list to an array. Use capabilities in the java.util package to write code to manipulate an array by sorting, performing a binary search, or converting the array to a list. Use the java.util.Comparator and java.lang.Comparable interfaces to affect the sorting of lists and arrays. Furthermore, recognize the effect of the "natural ordering" of primitive wrapper classes and java.lang.String on sorting.

Section 7: Fundamentals

7.1. Given a code example and a scenario, write code that uses the appropriate access modifiers, package declarations, and import statements to interact with (through access or inheritance) the code in the example.

7.2. Given an example of a class and a command-line, determine the expected runtime behavior.

7.3. Determine the effect upon object references and primitive values when they are passed into methods that perform assignments or other modifying operations on the parameters.

7.4. Given a code example, recognize the point at which an object becomes eligible for garbage collection, and determine what is and is not guaranteed by the garbage collection system. Recognize the behaviors of System.gc and finalization.

7.5. Given the fully-qualified name of a class that is deployed inside and/or outside a JAR file, construct the appropriate directory structure for that class. Given a code example and a classpath, determine whether the classpath will allow the code to compile successfully.

7.6. Write code that correctly applies the appropriate operators including assignment operators (limited to: =, + =, -=), arithmetic operators (limited to: +, -, *, /, %, ++, --), relational operators (limited to: <, < =, >, > =, = =, !=), the instanceof operator, logical operators (limited to: &, |, ^, !, &&, ||), and the conditional operator (? :), to produce a desired result. Write code that determines the equality of two objects or two primitives.