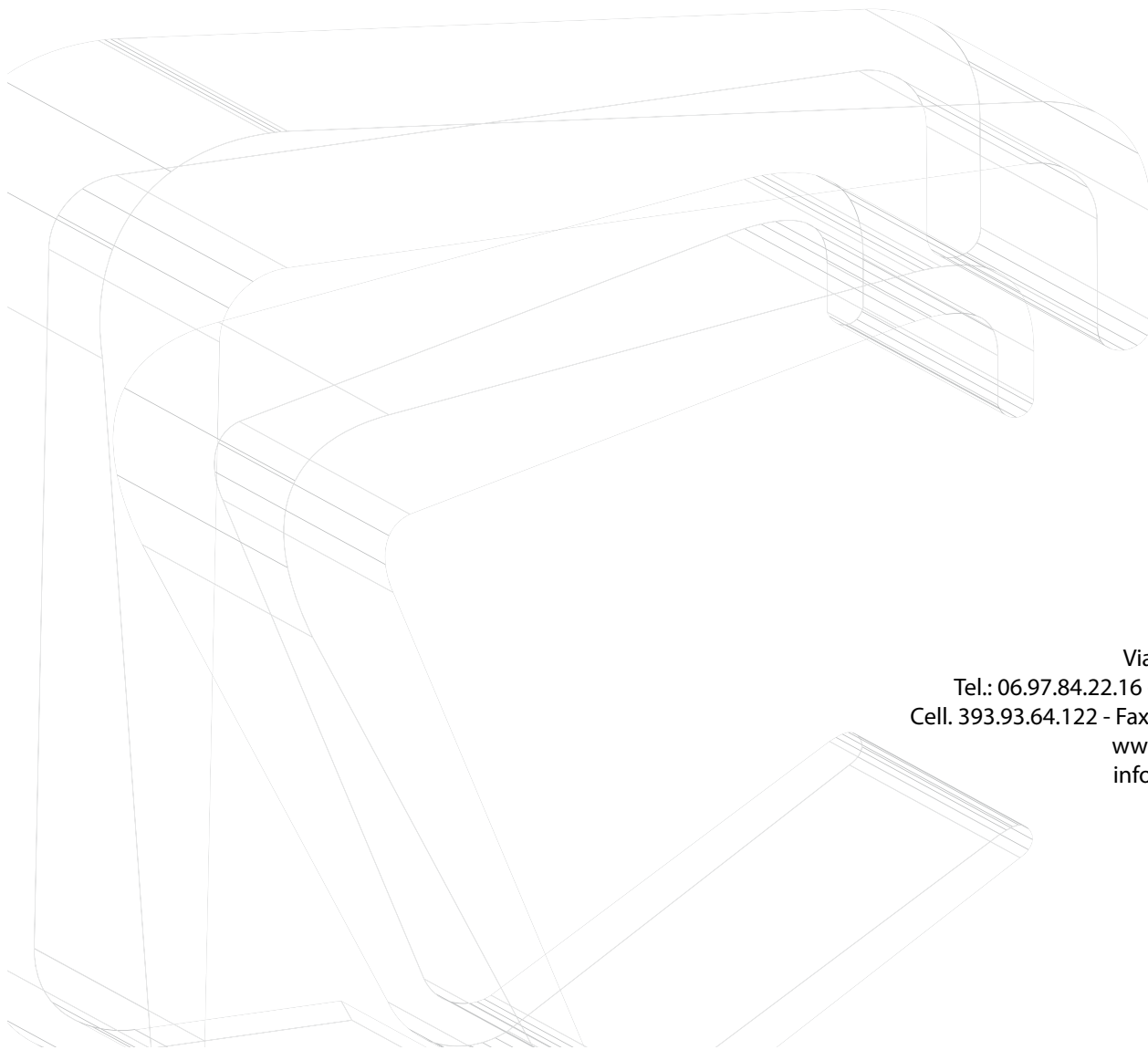




Programma corso Certificazione Java



PCAcademy
Via Capodistria 12
Tel.: 06.97.84.22.16 • 06.85.34.44.76
Cell. 393.93.64.122 - Fax: 06.91.65.92.92
www.pcademy.it
info@pcacademy.it

Informazioni generali

Questo corso è volto alla preparazione per l'esame Sun SCJP (Sun Certified Java Programmer).

Questa certificazione, riconosciuta in tutto il mondo, è una valida chiave di accesso al mondo del lavoro ed è alla base del verso le piattaforme J2SE e J2EE.

Il corso è quindi rivolto a chi della programmazione vuole fare una professione o a chi già opera nel campo dell'IT e vuole ottenere questa prestigiosa certificazione.

Il corso analizzerà punto per punto gli richiesti per il superamento dell'esame e fornirà tutti gli skills necessari.

Le lezioni teoriche saranno inoltre affiancate da test e simulazioni di prove di esame effettuate attraverso software specifico atte a monitorare i punti deboli del candidato.

Declarations, Initialization and Scoping

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).

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

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.

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.

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.

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.

Flow Control

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

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.

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

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

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.

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 programmatically.

API Contents

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.

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

Develop code that serializes and/or de-serializes objects using the following APIs from `java.io`: `DataInputStream`, `DataOutputStream`, `FileInputStream`, `FileOutputStream`, `ObjectInputStream`, `ObjectOutputStream` and `Serializable`.

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.

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/print` methods. Recognize and use formatting parameters (limited to: `%b`, `%c`, `%d`, `%f`, `%s`) in format strings.

Concurrency

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

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

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

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

OO Concepts

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

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.

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

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

Develop code that implements “is-a” and/or “has-a” relationships.

Collections / Generics

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.

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

Write code that uses the generic versions of the Collections API, in particular, the `Set`, `List`, 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. Write code that uses the `NavigableSet` and `NavigableMap` interfaces.

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.

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.

Fundamentals

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.

Given a code example, recognize the point at which an object becomes eligible for garbage collection, determine what is and is not guaranteed by the garbage collection system, and recognize the behaviors of the `Object.finalize()` method.

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.

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.