Feel free to contact santhosh_arena@yahoo.co.in

SCJP java Programmer certification Exam And Training. This site is entirely independent of Sun Microsystems Inc, The Sun Certified Java Programmers Exam (SCJP) is the internationally recognized java certification exam. I am offering free mock Exam preparation questions, practice tests, tutorials, certification faq and sample code. This page contains a very detailed tutorial organized by topic. At the end of the tutorial you can learn from your mistakes and apply your concepts on the free mock exams java certification practice tests.

Chapter 3 : Modifiers

1. Modifiers are Java keywords that provide information to compiler about the nature of the code, data and classes.

2. Access modifiers - public, protected, private

   a) Only applied to class level variables. Method variables are visible only inside the method.

   b) Can be applied to class itself (only to inner classes declared at class level, no such thing as protected or private top level class)

   c) Can be applied to methods and constructors.

   d) If a class is accessible, it doesn't mean, the members are also accessible. Members' accessibility determines what is accessible and what is not. But if the class is not accessible, the members are not accessible, even though they are declared public.

   e) If no access modifier is specified, then the accessibility is default package visibility. All classes in the same package can access the feature. It's called as friendly access. But friendly is not a Java keyword. Same directory is same package in Java's consideration.

   f) 'private' means only the class can access it, not even sub-classes. So, it'll cause access denial to a sub-class's own variable/method.

   g) These modifiers dictate, which classes can access the features. An instance of a class can access the private features of another instance of the same class.

   h) 'protected' means all classes in the same package (like default) and sub-classes in any package can access the features. But a subclass in another package can access the protected members in the super-class via only the references of subclass or its subclasses. A subclass in the same package doesn't have this restriction. This ensures that classes from other packages are accessing only the members that are part of their inheritance hierarchy.

   i) Methods cannot be overridden to be more private. Only the direction shown in following figure is permitted from parent classes to sub-classes.

      private → friendly (default) → protected → public

      Parent classes Sub-classes

3. final

a) final features cannot be changed.

b) The final modifier applies to classes, methods, and variables.

c) final classes cannot be sub-classed.

d) You can declare a variable in any scope to be final.

e) You may, if necessary, defer initialization of a final *local* variable. Simply declare the local variable and initialize it later (for final instance variables. You must initialize them at the time of declaration or in constructor).

f) final variables cannot be changed (result in a compile-time error if you do so)

g) final methods cannot be overridden.

h) Method arguments marked final are read-only. Compiler error, if trying to assign values to final arguments inside the method.

i) Member variables marked final are not initialized by default. They have to be explicitly assigned a value at declaration or in an initializer block. Static finals must be assigned to a value in a static initializer block, instance finals must be assigned a value in an instance initializer or in every constructor. Otherwise the compiler will complain.

j) A *blank final* is a final variable whose declaration lacks an initializer.

k) Final variables that are not assigned a value at the declaration and method arguments that are marked final are called blank final variables. They can be assigned a value at most once.

l) Local variables can be declared final as well.

m) If a final variable holds a reference to an object, then the state of the object may be changed by operations on the object, but the variable will always refer to the same object.

n) This applies also to arrays, because arrays are objects; if a final variable holds a reference to an array, then the components of the array may be changed by operations on the array, but the variable will always refer to the same array

o) A blank final instance variable must be definitely assigned at the end of every constructor of the class in which it is declared; otherwise a compile-time error occurs.

p) A class can be declared final if its definition is complete and no subclasses are desired or required.

q) A compile-time error occurs if the name of a final class appears in the extends clause of another class declaration; this implies that a final class cannot have any subclasses.

r) A compile-time error occurs if a class is declared both final and abstract, because the implementation of such a class could never be completed.

s) Because a final class never has any subclasses, the methods of a final class are never overridden

4. abstract

a) Can be applied to classes and methods.

b) For deferring implementation to sub-classes.

c) Opposite of final, final can't be sub-classed, abstract must be sub-classed.

d) A class should be declared abstract,

1. if it has any abstract methods.

2. if it doesn't provide implementation to any of the abstract methods it inherited

3. if it doesn't provide implementation to any of the methods in an interface that it says implementing.

e) Just terminate the abstract method signature with a ';', curly braces will give a compiler error.

f) A class can be abstract even if it doesn't have any abstract methods.

5. static

a) Can be applied to nested classes, methods, variables, free floating code-block (static initializer)

b) Static variables are initialized at class load time. A class has only one copy of these variables.

c) Static methods can access only static variables. (They have no this)

d) Access by class name is a recommended way to access static methods/variables.

e) Static initializer code is run at class load time.

f) Static methods may not be overridden to be non-static.

g) Non-static methods may not be overridden to be static.

h) Abstract methods may not be static.

i) Local variables cannot be declared as static.

j) Actually, static methods are not participating in the usual overriding mechanism of invoking the methods based on the class of the object at runtime. Static method binding is done at compile time, so the method to be invoked is determined by the type of reference variable rather than the actual type of the object it holds at runtime.

Let's say a sub-class has a static method which 'overrides' a static method in a parent class. If you have a reference variable of parent class type and you assign a child class object to that variable and invoke the static method, the method invoked will be the parent class method, not the child class method. The following code explains this.

```
public class StaticOverridingTest {
```

```
 public static void main(String s[]) {
        Child c = new Child();
        c.doStuff(); // This will invoke Child.doStuff()

        Parent p = new Parent();
        p.doStuff(); // This will invoke Parent.doStuff()
        p = c;
p.doStuff(); // This will invoke Parent.doStuff(), rather than Child.doStuff()
  }
}
class Parent {
  static int x = 100;
  public static void doStuff() {
        System.out.println("In Parent..doStuff");
        System.out.println(x);
  }
}
class Child extends Parent {
  static int x = 200;
  public static void doStuff() {
        System.out.println("In Child..doStuff");
        System.out.println(x);
  }
 }
```

6. **native**

a) Can be applied to methods only. (static methods also)

b) Written in a non-Java language, compiled for a single machine target type.

c) Java classes use lot of native methods for performance and for accessing hardware Java is not aware of.

d) Native method signature should be terminated by a ';', curly braces will provide a compiler error.

e) native doesn't affect access qualifiers. Native methods can be private.

f) Can pass/return Java objects from native methods.

g) System.loadLibrary is used in static initializer code to load native libraries. If the library is not loaded when the static method is called, an UnsatisfiedLinkError is thrown.

7. **transient**

a) Can be applied to class level variables only.(Local variables cannot be declared transient)

b) Transient variables may not be final or static.(But compiler allows the declaration, since it doesn't do any harm. Variables marked transient are never serialized. Static variables are not serialized anyway.)

c) Not stored as part of object's persistent state, i.e. not written out during serialization.

d) Can be used for security.

8. **synchronized**

a) Can be applied to methods or parts of methods only.

b) Used to control access to critical code in multi-threaded programs.

9. **volatile**

a) Can be applied to variables only.

b) Can be applied to static variables.

c) Cannot be applied to final variables.

d) Declaring a variable volatile indicates that it might be modified asynchronously, so that all threads will get the correct value of the variable.

e) Used in multi-processor environments.

Modifier	Class	Inner classes (Except local and anonymous classes)	Variable	Method		Constructor	Free floating Code block
public	Y	Y	Y	Y	Y		N
protected	N	Y	Y	Y	Y		N
(friendly) No access modifier	Y	Y (OK for all)	Y	Y	Y		N
private	N	Y	Y	Y	Y		N
Final	Y	Y (Except anonymous classes)	Y	Y	N		N
abstract	Y	Y (Except anonymous classes)	N	Y	N		N
Static	N	Y	Y	Y	N		Y (static initializer)
native	N	N	N	Y	N		N
transient	N	N	Y	N	N		N
Synchronized	N	N	N	Y	N		Y (part of method, also need to specify an object on which a lock should be obtained)
volatile	N	N	Y	N	N		N