Access and Non-Access

Modifiers in Java

Access Modifiers

public, private, protected, default

public – everywhere

private – only inside the class

protected – in the same package and in any derived class (sub-class)

default(no designation) – only in same package

Non-Access Modifiers

static, final, abstract, synchronized, transient, volatile, native, strictfp

abstract – can be a class, method or interface. An abstract method has no body and is typically implemented by a subclass. An abstract class typically has abstract methods and can't be instantiated. Prior to Java 8, all interfaces were purely abstract.

synchronized – method with this designation can't be accessed by multiple threads concurrently.

transient – applied to variables only and pertains to serialization. Basically, we don't save the value of the variable when the corresponding object is serialized. Useful for security.

volatile – indicates a variables value can be modified safely by different threads.

native – only for methods and indicates the usage of libraries and methods implemented in other programming languages.

strictfp – ensures calculations using floating-point precision are identical on all platforms.

final – can be used on classes, methods and variables.

final variable – can't be reassigned

```
class Demo{ final int MAX_GRADE  = 100;}  // can't change it anywhere now
```

```
class Demo(
    final int MAX_GRADE;   // if not initialized at declaration, all constructors must do so
    Demo(){ MAX_GRADE  = 100;}
}   // can call methods on final reference variables that change the state (list.add(4) etc.)
```

final class – can't be extended by another class(ex: String)

```
final class Student{}
class HSStudent extends Student{}   // won't compile
```

final method – subclass can't override

static – can be applied to classes, methods, interfaces, and variables.

static variables – think class variables since they belong to a class. In fact, they exist without any instances of the class. They are shared among all objects of the class and are not unique to any instance. It's considered poor programming style to access static variables from reference variables because it makes it appear they belong to an object when they don't. Please use the class dot notation like Math.PI.

```
Math m = null;
System.out.println(m.PI);   // illustrates that class variables do not belong to objects.
```

static methods – are methods at the class level and cannot access instance variables. Remember they exist prior to any objects ever being created. They can only access other static methods and variables (remember the reverse is true). Think of the utility methods from Collections, Arrays or the Math classes. They all manipulate or use the parameters to do some calculation.

static methods and variables are inherited by subclasses, but they are not involved in polymorphism or dynamic binding. That is, you can't override these methods, but you are permitted to redefine them.

static classes and interfaces – can't be the top-level class or interface. It can be used to designate inner classes or interfaces.

```
class Demo{
    static class InnerClass{}
    static interface InnerInterface{}
}
```