field. While the reference cannot be modified, the referenced object can be modified—with disastrous results.

Note that a nonzero-length array is always mutable, so it is nearly always wrong to have public static final array field. If a class has such a field, clients will be able to modify the contents of the array. This is a frequent source of security holes:

```java
// Potential security hole!
public static final Type[] VALUES = { ... };
```

The public array should be replaced by a private array and a public immutable list:

```java
private static final Type[] PRIVATE_VALUES = { ... };

public static final List VALUES =
    Collections.unmodifiableList(Arrays.asList(PRIVATE_VALUES));
```

Alternatively, if you require compile-time type safety and are willing to tolerate a performance loss, you can replace the public array field with a public method that returns a copy of a private array:

```java
private static final Type[] PRIVATE_VALUES = { ... };

public static final Type[] values() {
    return (Type[]) PRIVATE_VALUES.clone();
}
```

To summarize, you should always reduce accessibility as much as possible. After carefully designing a minimal public API, you should prevent any stray classes, interfaces, or members from becoming a part of the API. With the exception of public static final fields, public classes should have no public fields. Ensure that objects referenced by public static final fields are immutable.