DCL01-J. Do not reuse public identifiers from the Java Standard Library

Do not reuse the names of publicly visible identifiers, public utility classes, interfaces, or packages in the Java Standard Library.

When a developer uses an identifier that has the same name as a public class, such as `Vector`, a subsequent maintainer might be unaware that this identifier does not actually refer to `java.util.Vector` and might unintentionally use the custom `Vector` rather than the original `java.util.Vector` class. The custom type `Vector` can shadow a class name from `java.util.Vector`, as specified by *The Java Language Specification* (JLS), §6.3.2, "Obscured Declarations" [JLS 2005], and unexpected program behavior can occur.

Well-defined import statements can resolve these issues. However, when reused name definitions are imported from other packages, use of the *type-import-on-demand declaration* (see §7.5.2, "Type-Import-on-Demand Declaration" [JLS 2005]) can complicate a programmer's attempt to determine which specific definition was intended to be used. Additionally, a common practice that can lead to errors is to produce the import statements *after* writing the code, often via automatic inclusion of import statements by an IDE, which creates further ambiguity with respect to the names. When a custom type is found earlier than the intended type in the Java include path, no further searches are conducted. Consequently, the wrong type is silently adopted.

Noncompliant Code Example (Class Name)

This noncompliant code example implements a class that reuses the name of the class `java.util.Vector`. It attempts to introduce a different condition for the `isEmpty()` method for interfacing with native legacy code by overriding the corresponding method in `java.util.Vector`. Unexpected behavior can arise if a maintainer confuses the `isEmpty()` method with the `java.util.Vector.isEmpty()` method.

```
class Vector {
  private int val = 1;

  public boolean isEmpty() {
    if (val == 1) {   // Compares with 1 instead of 0
      return true;
    } else {
      return false;
    }
  }
  // Other functionality is same as java.util.Vector
}

// import java.util.Vector; omitted
public class VectorUser {
  public static void main(String[] args) {
    Vector v = new Vector();
    if (v.isEmpty()) {
      System.out.println("Vector is empty");
    }
  }
}
```

Compliant Solution (Class Name)

This compliant solution uses a different name for the class, preventing any potential shadowing of the class from the Java Standard Library:

```
class MyVector {
  //Other code
}
```

When the developer and organization control the original shadowed class, it may be preferable to change the design strategy of the original in accordance with Bloch's *Effective Java* [Bloch 2008], Item 16, "Prefer Interfaces to Abstract Classes." Changing the original class into an interface would permit class `MyVector` to declare that it implements the hypothetical `Vector` interface. With this technique, client code that intended to use `MyVector` would remain compatible with code that uses the original implementation of `Vector`.

Risk Assessment

Public identifier reuse decreases the readability and maintainability of code.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
DCL01-J	Low	Unlikely	Medium	**P2**	**L3**

Automated Detection

An automated tool can easily detect reuse of the set of names representing public classes or interfaces from the Java Standard Library.

Related Guidelines

SEI CERT C Coding Standard	PRE04-C. Do not reuse a standard header file name
SEI CERT C++ Coding Standard	VOID PRE04-CPP. Do not reuse a standard header file name

Bibliography

[Bloch 2005]	Puzzle 67, "All Strung Out"
[Bloch 2008]	Item 16, "Prefer Interfaces to Abstract Classes"
[FindBugs 2008]	
[JLS 2005]	§6.3.2, "Obscured Declarations" §6.3.1, "Shadowing Declarations" §7.5.2, "Type-Import-on-Demand Declaration" §14.4.3, "Shadowing of Names by Local Variables"