EJERCICIO 1:

VULNERABILIDADES:

-**Insecure temporary file creation methods should not be used**. Los archivos temporales se consideran creados de forma insegura cuando la verificación de la existencia del archivo se realiza por separado de la creación del archivo real. Esta situación puede ocurrir cuando se crean archivos temporales usando funciones normales de manejo de archivos o cuando se usan funciones dedicadas de manejo de archivos temporales que no son atómicas.

-**Authorizations should be based on strong decisions.** Al otorgar a los usuarios acceso a los recursos de una aplicación, dicha autorización debe basarse en decisiones sólidas. Por ejemplo, un usuario puede estar autorizado a acceder a un recurso solo si está autenticado o si tiene la función y los privilegios correctos.

- **Logging should not be vulnerable to injection attacks.** La inyección de registros ocurre cuando una aplicación no logra desinfectar los datos que no son de confianza utilizados para el registro. Un atacante puede falsificar el contenido del registro para evitar que una organización pueda rastrear actividades maliciosas.

BUGS:

-**Recursion should not be infinite.** Los métodos recursivos siempre deben llegar a un caso base en el que se detenga la recursividad. La recursividad infinita es causada por errores lógicos y bloqueará su aplicación.

-**Loops should not be infinite.** Un bucle infinito nunca terminará mientras se ejecuta el programa, lo que significa que debe finalizar el programa para salir del bucle. Cada bucle debe tener una condición final, ya sea cumpliendo la condición de terminación del bucle o mediante una declaración de interrupción.

-**Exceptions should not be created without being thrown.** Crear un nuevo Throwable sin tirarlo realmente es inútil y probablemente se deba a un error.

SECURITY HOTSPOT:

-**Allowing both safe and unsafe HTTP methods is security-sensitive.** Un método HTTP es seguro cuando se utiliza para realizar una operación de solo lectura, como recuperar información. Por el contrario, se utiliza un método HTTP inseguro para cambiar el estado de una aplicación, por ejemplo, para actualizar el perfil de un usuario en una aplicación web. Los métodos HTTP seguros comunes son GET, HEAD u OPTIONS. Los métodos HTTP inseguros comunes son POST, PUT y DELETE. Permitir que métodos HTTP seguros e inseguros realicen una operación específica en una aplicación web podría afectar su seguridad; por ejemplo, las protecciones CSRF la mayoría de las veces solo protegen operaciones realizadas por métodos HTTP inseguros.

-**Formatting SQL queries is security-sensitive.** Las consultas SQL formateadas pueden ser difíciles de mantener y depurar y pueden aumentar el riesgo de inyección de SQL al concatenar valores que no son de confianza en la consulta. Sin embargo, esta regla no detecta inyecciones de SQL (a diferencia de la regla S3649), el objetivo es solo resaltar consultas complejas/formateadas.

-**Using clear-text protocols is security-sensitive.** Los protocolos de texto claro como ftp, telnet o http carecen de cifrado de los datos transportados, así como de la capacidad de crear una conexión autenticada. Significa que un atacante capaz de detectar el tráfico de la red puede leer, modificar o corromper el contenido transportado. Estos protocolos no son seguros ya que exponen las aplicaciones a una amplia gama de riesgos: exposición de datos sensibles, tráfico redirigido a un punto final malicioso, actualización o instalador de software infectado con malware, ejecución de código del lado del cliente,corrupción de información crítica.Incluso en el contexto de redes aisladas, como entornos fuera de línea o entornos de nube segmentados, la amenaza interna existe. Por lo tanto, aún pueden ocurrir ataques que impliquen el rastreo o manipulación de comunicaciones.

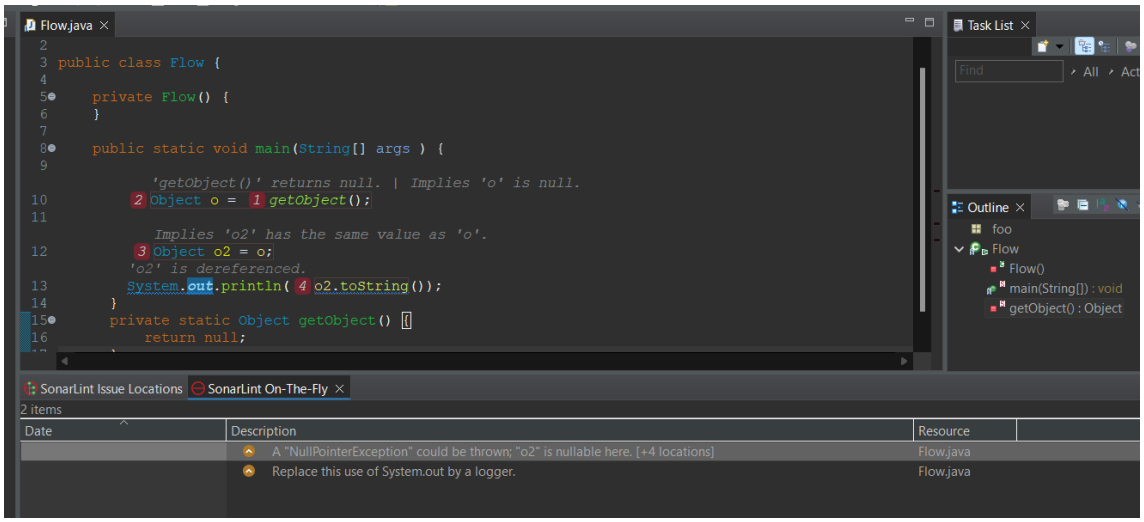
CODE SMELL:

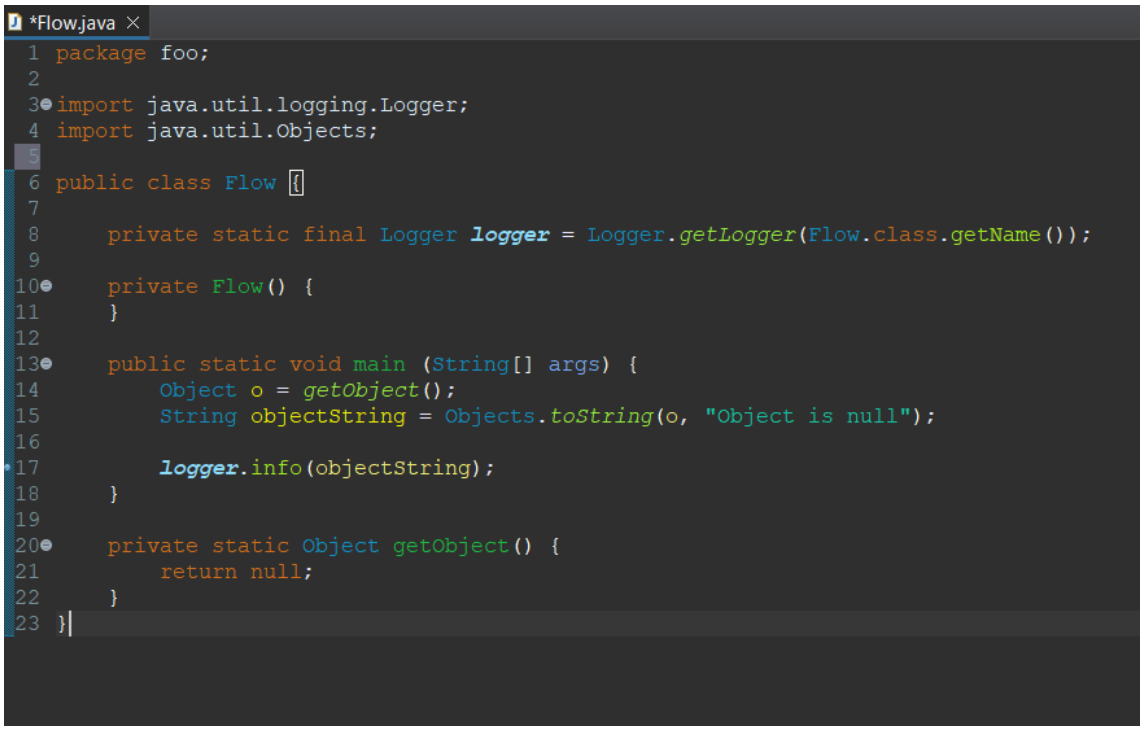
-Conditionals should start on new lines. Colocar una declaración if en la misma línea que el cierre } de un bloque if, else o else if anterior puede generar confusión y posibles errores. Puede indicar que falta una declaración else o crear ambigüedad para los mantenedores que podrían no comprender que las dos declaraciones no están conectadas.

-"NullPointerException" should not be caught. NullPointerException debe evitarse, no detectarse. Cualquier situación en la que se capture explícitamente NullPointerException se puede convertir fácilmente en una prueba nula, y cualquier comportamiento que se lleve a cabo en el bloque catch se puede mover fácilmente a la rama "es nulo" del condicional.

-public static" fields should be constant. No hay ninguna buena razón para declarar un campo "público" y "estático" sin declararlo también "final". La mayoría de las veces se trata de una chapuza al compartir un estado entre varios objetos. Pero con este enfoque, cualquier objeto puede hacer lo que quiera con el estado compartido, como establecerlo en nulo.

EJERCICIO 2


Flow.java
public class Flow {
private Flow() {
}
public static void main(String[] args) {
'getObject()' returns null. | Implies 'o' is null.
2 Object o = 1 getObject();
Implies 'o2' has the same value as 'o'.
3 Object o2 = o;
'o2' is dereferenced.
System.out.println(4 o2.toString());
}
private static Object getObject() {
return null;
Task List
Find
All
Act
Outline
foo
Flow
Flow()
main(String[]) : void
getObject() : Object
SonarLint Issue Locations
SonarLint On-The-Fly
2 items
Date
Description
Resource
A "NullPointerException" could be thrown; "o2" is nullable here. [+4 locations]
Flow.java
Replace this use of System.out by a logger.
Flow.java


*Flow.java
1 package foo;
2
3 import java.util.logging.Logger;
4 import java.util.Objects;
5
6 public class Flow {
7
8 private static final Logger logger = Logger.getLogger(Flow.class.getName());
9
10 private Flow() {
11 }
12
13 public static void main (String[] args) {
14 Object o = getObject();
15 String objectString = Objects.toString(o, "Object is null");
16
17 logger.info(objectString);
18 }
19
20 private static Object getObject() {
21 return null;
22 }
23 }