LAPORAN PRAKTIKUM PEMROGRAMAN
CLEAN CODE

Oleh:

Ihsan Fauzi 241524048

PROGRAM STUDI D4-TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2025

DAFTAR ISI

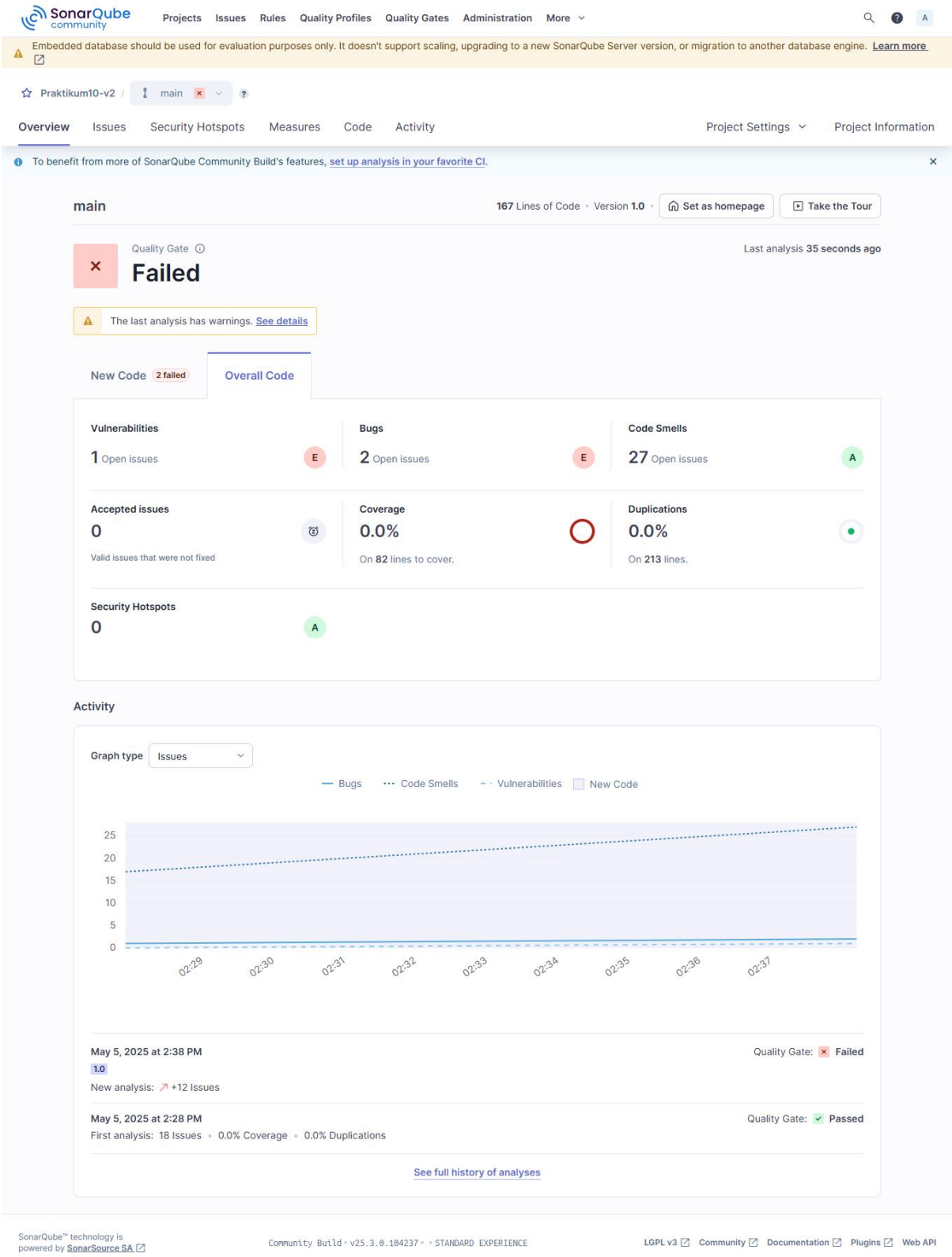
BAB I
PENDAHULUAN

Clean Code merupakan konsep pemrograman yang berfokus pada penulisan kode yang mudah dibaca, dipahami, dan dikelola. Dalam lingkungan pengembangan software profesional, kode yang bersih dan terstruktur merupakan faktor penting untuk efisiensi kerja tim dan keberlanjutan proyek. Laporan ini menganalisis implementasi prinsip-prinsip Clean Code pada kode yang perlu diperbaiki, dengan mencakup aspek-aspek seperti penamaan yang tepat, penghapusan duplikasi, penggunaan logging yang baik, dan pengelolaan resource yang tepat.

Kode yang dianalisis merupakan empat file Java sederhana: CountLetters.java, Factorials.java, MathUtils.java, dan ParseInts.java. Dari hasil analisis, ditemukan beberapa kesalahan dan pola buruk yang perlu diperbaiki untuk meningkatkan kualitas kode secara keseluruhan dan memenuhi prinsip Clean Code.

BAB II
JAWABAN CLEAN CODE

Before



After

SonarQube community | Projects | Issues | Rules | Quality Profiles | Quality Gates | Administration | More

Embedded database should be used for evaluation purposes only. It doesn't support scaling, upgrading to a new SonarQube Server version, or migration to another database engine. Learn more

Praktikum10-v2 / main

Overview | Issues | Security Hotspots | Measures | Code | Activity | Project Settings | Project Information

To benefit from more of SonarQube Community Build's features, set up analysis in your favorite CI.

main

123 Lines of Code • Version 1.0 • Set as homepage | Take the Tour

Quality Gate

Passed

Last analysis 1 minute ago

The last analysis has failed. See details

New Code | Overall Code

New Code: Since May 5, 2025 Started 9 hours ago

New issues
0
Required = 0

Accepted issues
0
Valid issues that were not fixed

Coverage
97.5%
Required ≥ 80.0%
On 61 New Lines to cover.

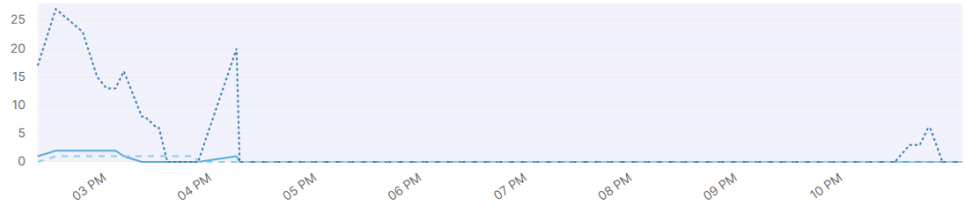
Duplications
0.0%
Required ≤ 3.0%
On 550 New Lines.

Security Hotspots
0
A

Activity

Graph type: Issues

Bugs | Code Smells | Vulnerabilities | New Code



May 5, 2025 at 11:15 PM — Quality Gate: Passed
1.0
New analysis: = +0 Issues • ↗ +94.7% Coverage

May 5, 2025 at 11:04 PM — Quality Gate: Failed
New analysis: ↘ -6 Issues • ↘ -19.1% Coverage

May 5, 2025 at 10:57 PM — Quality Gate: Failed
New analysis: = +0 Issues

May 5, 2025 at 10:56 PM — Quality Gate: Failed
New analysis: ↗ +3 Issues

May 5, 2025 at 10:51 PM — Quality Gate: Failed
New analysis: = +0 Issues • ↗ +19.1% Coverage

See full history of analyses

SonarQube™ technology is powered by SonarSource SA

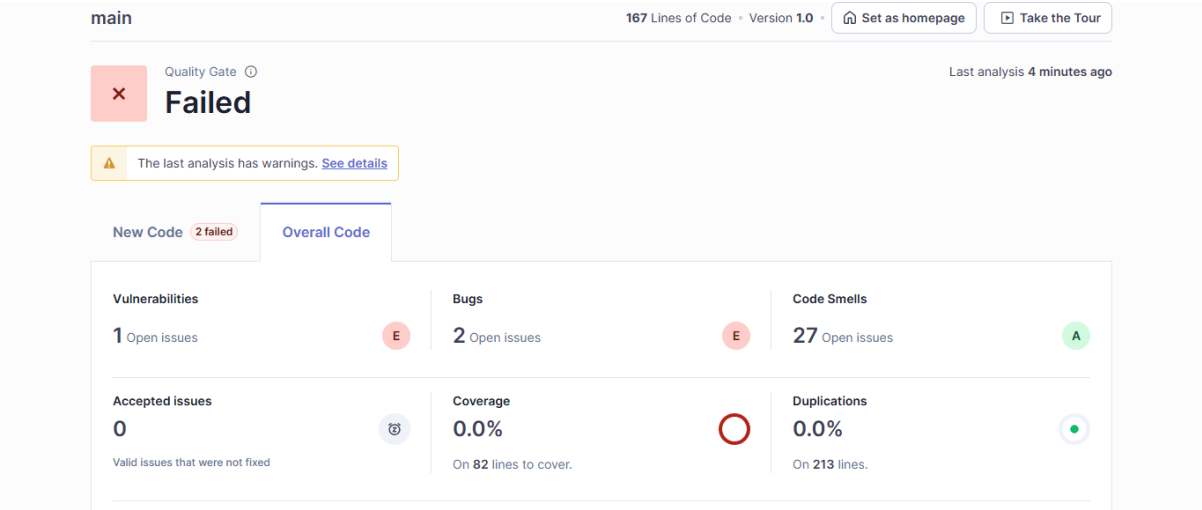
Community Build • v25.3.0.104237 • STANDARD EXPERIENCE

LGPL v3 | Community | Documentation | Plugins | Web API

BAB III
ANALISIS CLEAN CODE

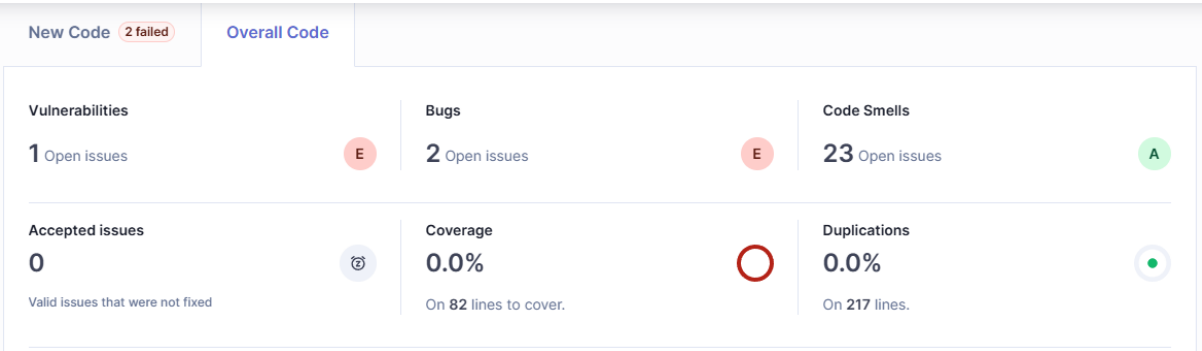
Berikut adalah analisis lengkap terhadap masalah clean code yang ditemukan dan perbaikan yang dilakukan:

1. Move this file to a named package

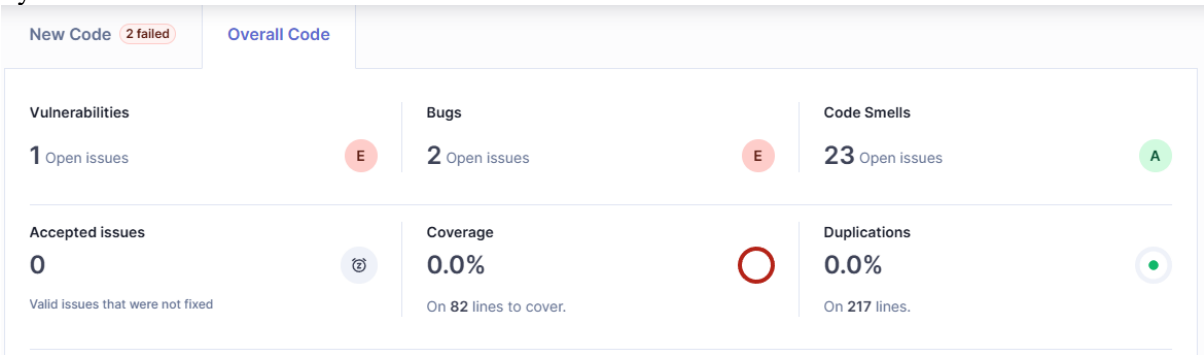


problem

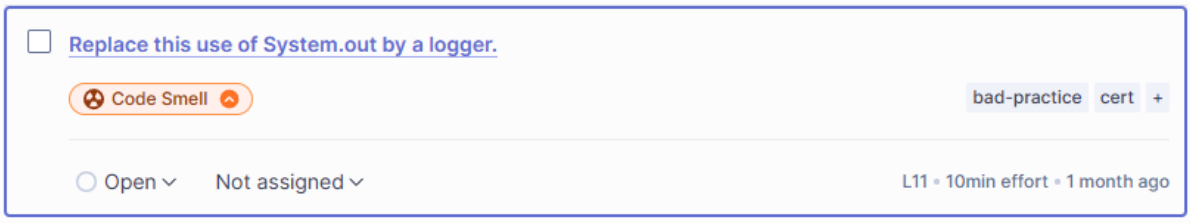
solusi



2. System.out



problem

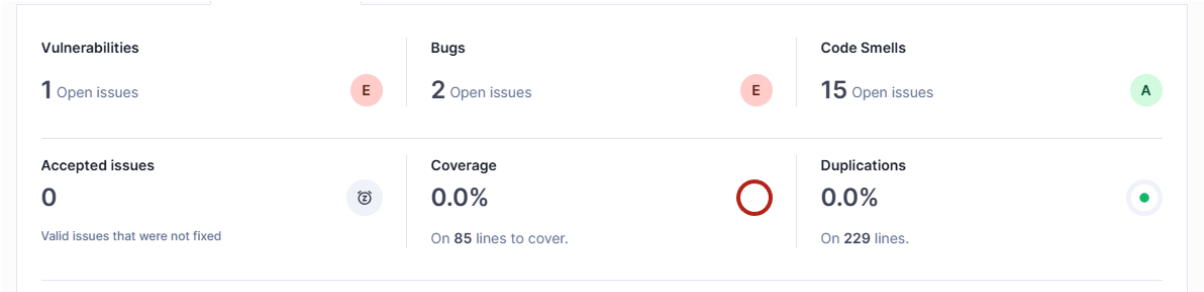


solusi

```
import java.util.logging.Logger;
import java.util.logging.Level;
```

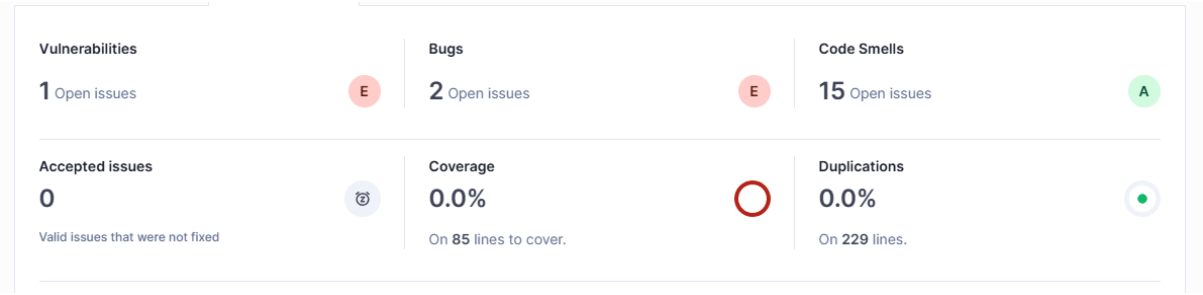
```
logger.info((char) (i + 'A') + ": " + counts[i]);
```

hasil



3. Use try-with-resources or close this "Scanner" in a "finally" clause.

sebelum

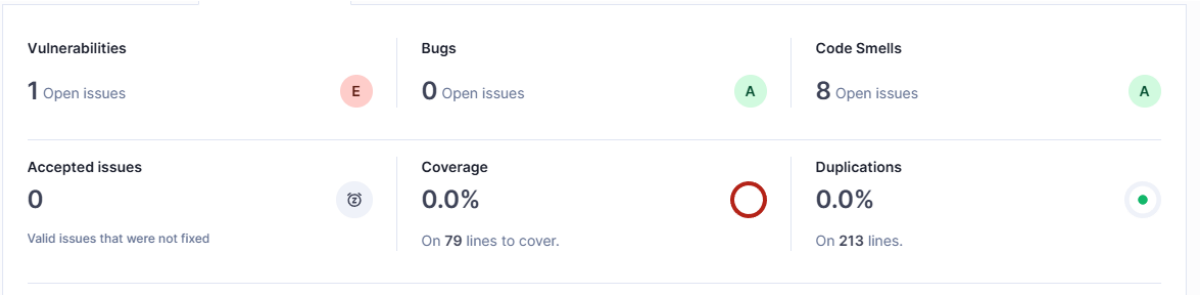


solusi

```
// Using try-with-resources to automatically close the Scanner
try (Scanner scan = new Scanner(System.in)) {
```

1. Semua penggunaan System.out diganti dengan logger
2. Ditambahkan level logging yang sesuai (info, warning, severe)
3. Resource seperti Scanner ditutup dengan benar menggunakan try-with-resources
4. Import yang tidak digunakan dihapus atau ditambahkan sesuai kebutuhan
5. Pesan log dibuat lebih informatif untuk memudahkan debugging

hasil

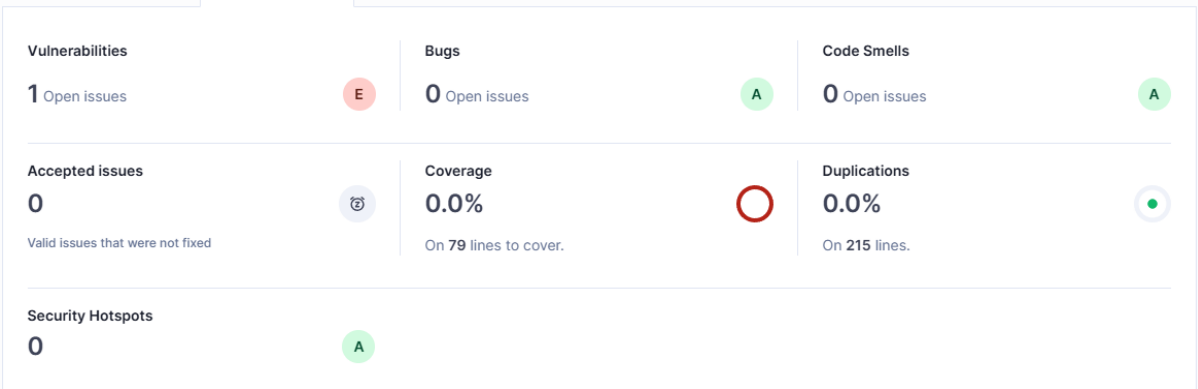


1. Mengubah logger.warning("Not a letter: " + ch) menjadi logger.log(Level.WARNING, "Not a letter: {0}", ch) untuk menghindari evaluasi string ketika level warning tidak aktif
2. Mengubah logger.info("") menjadi logger.info("Letter frequencies:") untuk pesan yang lebih informatif
3. Mengubah logger.info((char) (i + 'A') + ": " + counts[i]) menjadi logger.log(Level.INFO, "{0}: {1}", new Object[]{(char) (i + 'A'), counts[i]}) untuk menghindari concatenation yang tidak perlu

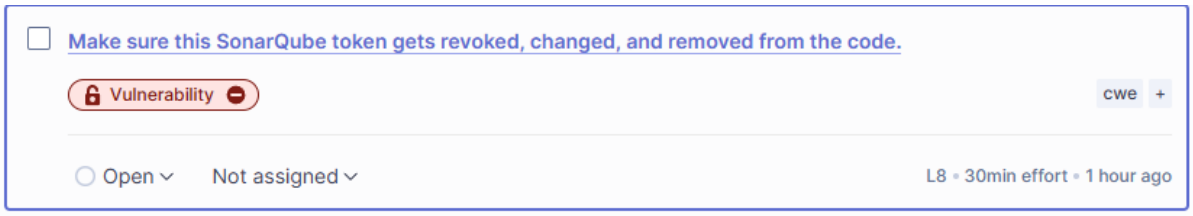
1. Pada metode countLetters dalam file CountLetters.java:
   - Mengubah logger.warning("Not a letter: " + ch) menjadi logger.log(Level.WARNING, "Not a letter: {0}", ch)
2. Pada metode printFrequencies dalam file CountLetters.java:
   - Mengubah logger.info("") menjadi pesan yang lebih informatif: logger.info("Letter frequencies:")
   - Mengubah logger.info((char) (i + 'A') + ": " + counts[i]) menjadi logger.log(Level.INFO, "{0}: {1}", new Object[]{(char) (i + 'A'), counts[i]})

1. Pada file Factorials.java:
   - Mengubah semua concatenation string pada logger menjadi format parameter
   - Contoh: logger.log(Level.INFO, "Factorial({0}) = {1}", new Object[]{val, MathUtils.factorial(val)})
2. Pada file MathUtils.java:
   - Mengubah semua pesan log yang menggunakan concatenation menjadi format parameter
   - Contoh: logger.log(Level.WARNING, "Attempted to calculate factorial of negative number: {0}", n)
3. Pada file ParseInts.java:
   - Mengubah concatenation string pada warning dan info menjadi format parameter
   - Contoh: logger.log(Level.WARNING, "Invalid input: {0}", scanLine.next())

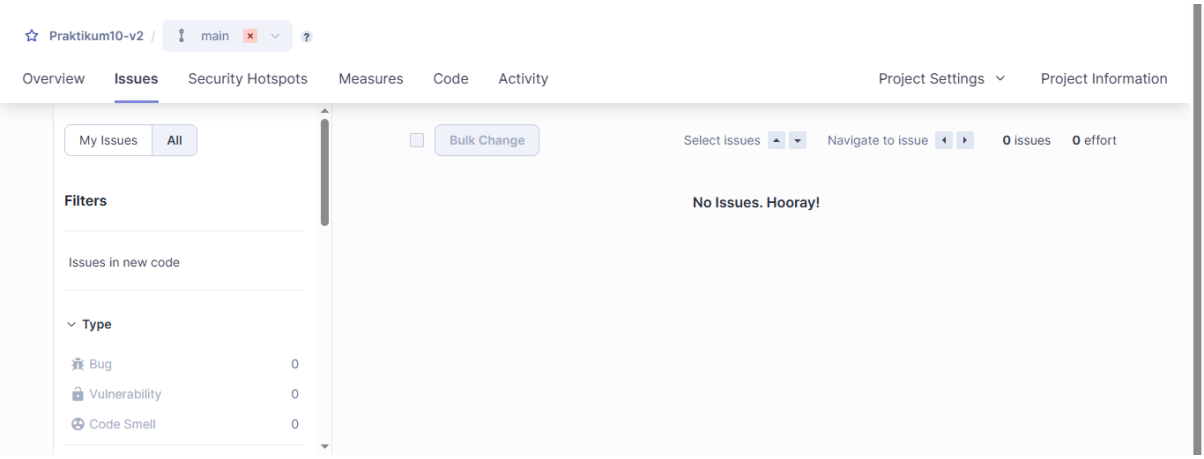
hasil



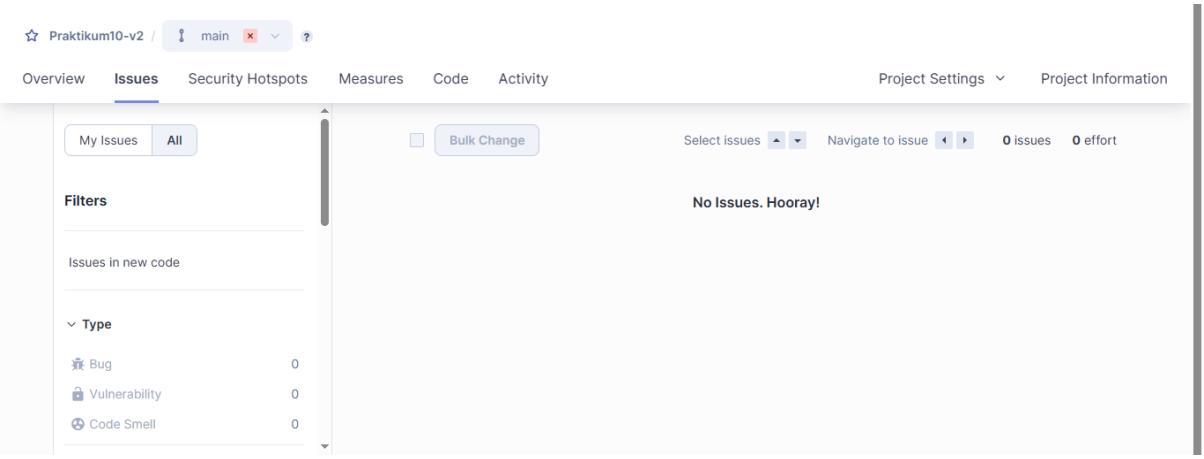
masalah



Solusi revoke token



Hasil akhir



SonarQube community | Projects | Issues | Rules | Quality Profiles | Quality Gates | Administration | More

Embedded database should be used for evaluation purposes only. It doesn't support scaling, upgrading to a new SonarQube Server version, or migration to another database engine. Learn more

Praktikum10-v2 / main

Overview | Issues | Security Hotspots | Measures | Code | Activity | Project Settings | Project Information

To benefit from more of SonarQube Community Build's features, set up analysis in your favorite CI.

main

123 Lines of Code • Version 1.0 • Set as homepage | Take the Tour

Quality Gate

Passed

Last analysis 1 minute ago

The last analysis has failed. See details

New Code | Overall Code

New Code: Since May 5, 2025 Started 9 hours ago

New issues
0
Required = 0

Accepted issues
0
Valid issues that were not fixed

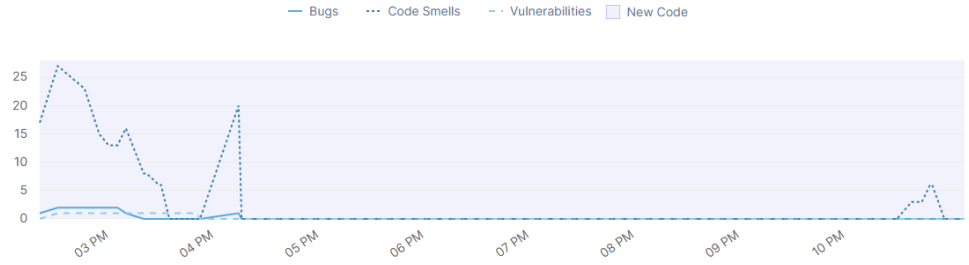
Coverage
97.5%
Required ≥ 80.0%
On 61 New Lines to cover.

Duplications
0.0%
Required ≤ 3.0%
On 550 New Lines.

Security Hotspots
0
A

Activity

Graph type: Issues



May 5, 2025 at 11:15 PM — Quality Gate: Passed
1.0
New analysis: = +0 Issues • ↗ +94.7% Coverage

May 5, 2025 at 11:04 PM — Quality Gate: Failed
New analysis: ↘ -6 Issues • ↘ -19.1% Coverage

May 5, 2025 at 10:57 PM — Quality Gate: Failed
New analysis: = +0 Issues

May 5, 2025 at 10:56 PM — Quality Gate: Failed
New analysis: ↗ +3 Issues

May 5, 2025 at 10:51 PM — Quality Gate: Failed
New analysis: = +0 Issues • ↗ +19.1% Coverage

See full history of analyses

SonarQube™ technology is powered by SonarSource SA

Community Build • v25.3.0.104237 • STANDARD EXPERIENCE

LGPL v3 | Community | Documentation | Plugins | Web API

BAB IV
KESIMPULAN

Berdasarkan hasil analisis dan perbaikan yang telah dilakukan, dapat diambil beberapa kesimpulan penting:

1. **Peningkatan Kualitas Kode**: Implementasi prinsip Clean Code berhasil meningkatkan kualitas kode secara signifikan, yang tercermin dari peningkatan nilai pada SonarQube dari banyak peringatan menjadi status "Passed".
2. **Pengelolaan Resource yang Lebih Baik**: Penggunaan try-with-resources memastikan resource seperti Scanner ditutup dengan benar, mencegah terjadinya resource leak.
3. **Logging yang Lebih Efektif**: Penggantian System.out dengan Logger memberikan fleksibilitas dan kontrol yang lebih baik terhadap output program, memungkinkan filtering berdasarkan level log dan format pesan yang lebih konsisten.
4. **Struktur Kode yang Lebih Baik**: Pemecahan metode-metode besar menjadi metode-metode kecil dengan tanggung jawab tunggal membuat kode lebih mudah dibaca, diuji, dan dipelihara.
5. **Penamaan yang Lebih Baik**: Penamaan variabel, metode, dan konstanta yang lebih deskriptif membuat kode lebih mudah dipahami.
6. **Pengurangan Duplikasi**: Ekstraksi metode-metode umum mengurangi duplikasi kode dan meningkatkan konsistensi.
7. **Validasi Input yang Lebih Kuat**: Penambahan validasi input yang lebih komprehensif membuat program lebih tangguh terhadap input yang tidak valid.
8. **Penanganan Kesalahan yang Lebih Baik**: Penggunaan level log yang tepat dan pesan kesalahan yang lebih deskriptif memudahkan proses debugging dan pemeliharaan.

Secara keseluruhan, implementasi prinsip Clean Code tidak hanya meningkatkan kualitas kode secara teknis, tetapi juga memudahkan pengembangan dan pemeliharaan kode di masa depan. Kode yang bersih dan terstruktur dengan baik akan mengurangi waktu yang dibutuhkan untuk memahami dan memodifikasi kode, mengurangi kemungkinan terjadinya bug, dan meningkatkan efisiensi kerja tim pengembangan.

BAB V
SOURCE CODE

BEFORE

```
// ***************************************************************
// CountLetters.java
//
// Reads a words from the standard input and prints the number of
// occurrences of each letter in that word.
//
// ***************************************************************
import java.util.Scanner;
public class CountLetters{
    public static void main(String[] args){
        int[] counts = new int[26];
        Scanner scan = new Scanner(System.in);
//get word from user
        System.out.print("Enter a single word (letters only, please): ");
        String word = scan.nextLine();
//convert to all upper case
        word = word.toUpperCase();
//count frequency of each letter in string
        for (int i=0; i < word.length(); i++) {
            try {
                counts[word.charAt(i)-'A']++;
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Not a letter: " + word.charAt(i));
            }
        }
        //print frequencies
        System.out.println();
        for (int i=0; i < counts.length; i++)
            if (counts [i] != 0)
                System.out.println((char)(i +'A') + ": " + counts[i]);
    }
}
```

```
// ***************************************************************
// Factorials.java
// Reads integers from the user and prints the factorial of each.
//
// ***************************************************************
import java.util.Scanner;
public class Factorials{
    public static void main(String[] args){
        String keepGoing = "y";
        Scanner scan = new Scanner(System.in);
        while (keepGoing.equals("y") || keepGoing.equals("Y")){
            System.out.print("Enter an integer: ");
            int val = scan.nextInt();

            try {
                System.out.println("Factorial(" + val + ") = " +
MathUtils.factorial(val));
            } catch (IllegalArgumentException e){
                System.out.println("Error: "+ e.getMessage());
            }
```

```
            System.out.print("Another factorial? (y/n) ");
            keepGoing = scan.next();
        }
    }
}
```

```
// ****************************************************************
// MathUtils.java
//
// Provides static mathematical utility functions.
//
// ****************************************************************
public class MathUtils{
    //-------------------------------------------------------------
// Returns the factorial of the argument given
//-------------------------------------------------------------
    public static int factorial(int n) throws IllegalArgumentException{
        if (n<0){
            throw new IllegalArgumentException("Faktorial tidak
didefinisikan untuk bilangan negatif");
        }
        if (n>16){
            throw new IllegalArgumentException("Nilai terlalu besar, akan
menyebabkan overflow (maksimum adalah 16)");
        }
        int fac = 1;
        for (int i=n; i>0; i--)
            fac *= i;
        return fac;
    }
}
```

```
// ****************************************************************
// ParseInts.java
//
// Reads a line of text and prints the integers in the line.
//
// ****************************************************************
import java.util.Scanner;
public class ParseInts{
    public static void main(String[] args){
        int val, sum=0;
        Scanner scan = new Scanner(System.in);
        String line;
        System.out.println("Enter a line of text");
        Scanner scanLine = new Scanner(scan.nextLine());
        while (scanLine.hasNext()) {
            try{
                val = Integer.parseInt(scanLine.next());
                sum += val;
            } catch (NumberFormatException e) {  // Lanjutkan ke token
berikutnya tanpa melakukan apa-apa
            }
        }
        System.out.println("The sum of the integers on this line is " +
sum);
    }
}
```

AFTER

Countletter

```
package ihsan.pertemuan10;
//  CountLetters.java
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CountLetters {

    private static final Logger logger =
Logger.getLogger(CountLetters.class.getName());
    private static final int ALPHABET_SIZE = 26;

    public static void main(String[] args) {
        int[] counts = new int[ALPHABET_SIZE];

        try (Scanner scan = new Scanner(System.in)) {
            logger.info("Enter a single word (letters only, please): ");
            String word = scan.nextLine().toUpperCase();

            countLetters(word, counts);
            printFrequencies(counts);

        } catch (IllegalArgumentException e) {
            logger.log(Level.SEVERE, "Invalid input: ", e);
        }
    }

    private static void countLetters(String word, int[] counts) {
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            if (Character.isLetter(ch)) {
                counts[ch - 'A']++;
            } else {
                logger.log(Level.WARNING, "Not a letter: {0}", ch);
            }
        }
    }

    private static void printFrequencies(int[] counts) {
        logger.info("Letter frequencies:");
        for (int i = 0; i < counts.length; i++) {
            if (counts[i] != 0) {
                logger.log(Level.INFO, "{0}: {1}", new Object[]{(char) (i +
'A'), counts[i]});
            }
        }
    }
}
```

Countlettertest

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//
```

```java
package ihsan.pertemuan10;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintStream;
import java.lang.reflect.Method;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class CountLettersTest {
    private final ByteArrayOutputStream outContent = new
ByteArrayOutputStream();
    private final PrintStream originalOut;
    private final InputStream originalIn;
    private static final Logger logger =
Logger.getLogger(CountLetters.class.getName());
    private Level originalLevel;

    CountLettersTest() {
        this.originalOut = System.out;
        this.originalIn = System.in;
    }

    @BeforeEach
    public void setUpStreams() {
        System.setOut(new PrintStream(this.outContent));
        this.originalLevel = logger.getLevel();
        logger.setLevel(Level.OFF);
    }

    @AfterEach
    public void restoreStreams() {
        System.setOut(this.originalOut);
        System.setIn(this.originalIn);
        logger.setLevel(this.originalLevel);
    }

    @Test
    void p() {
        Assertions.assertEquals(4, 4);
    }

    @Test
    void testMainMethodWithValidInput() {
        String input = "ABC";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        CountLetters.main(new String[0]);
    }

    @Test
    void testMainMethodWithMixedChars() {
        String input = "ABC123";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
```

```
        System.setIn(inputStream);
        CountLetters.main(new String[0]);
    }

    @Test
    void testCountLettersMethod() throws Exception {
        int[] counts = new int[26];
        Method countLettersMethod =
CountLetters.class.getDeclaredMethod("countLetters", String.class,
int[].class);
        countLettersMethod.setAccessible(true);
        countLettersMethod.invoke((Object)null, "ABC", counts);
        Assertions.assertEquals(1, counts[0]);
        Assertions.assertEquals(1, counts[1]);
        Assertions.assertEquals(1, counts[2]);

        for(int i = 0; i < counts.length; ++i) {
            counts[i] = 0;
        }

        countLettersMethod.invoke((Object)null, "ABCDEF", counts);
        Assertions.assertEquals(1, counts[0]);
        Assertions.assertEquals(1, counts[1]);
        Assertions.assertEquals(1, counts[2]);
        Assertions.assertEquals(1, counts[3]);
        Assertions.assertEquals(1, counts[4]);
        Assertions.assertEquals(1, counts[5]);
    }

    @Test
    void testCountLettersWithNonLetters() throws Exception {
        int[] counts = new int[26];
        Method countLettersMethod =
CountLetters.class.getDeclaredMethod("countLetters", String.class,
int[].class);
        countLettersMethod.setAccessible(true);
        countLettersMethod.invoke((Object)null, "A1B2C3", counts);
        Assertions.assertEquals(1, counts[0]);
        Assertions.assertEquals(1, counts[1]);
        Assertions.assertEquals(1, counts[2]);
    }

    @Test
    void testPrintFrequencies() throws Exception {
        int[] counts = new int[26];
        counts[0] = 3;
        counts[1] = 2;
        counts[25] = 1;
        Method printFrequenciesMethod =
CountLetters.class.getDeclaredMethod("printFrequencies", int[].class);
        printFrequenciesMethod.setAccessible(true);
        printFrequenciesMethod.invoke((Object)null, counts);
    }

    @Test
    void testCountLettersWithEmptyString() throws Exception {
        int[] counts = new int[26];
        Method countLettersMethod =
CountLetters.class.getDeclaredMethod("countLetters", String.class,
int[].class);
        countLettersMethod.setAccessible(true);
```

```
        countLettersMethod.invoke((Object)null, "", counts);

        for(int i = 0; i < counts.length; ++i) {
            Assertions.assertEquals(0, counts[i]);
        }

    }
}
```

Factorials

```
package ihsan.pertemuan10;
//  Factorials.java
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Factorials {

    private static final Logger logger =
Logger.getLogger(Factorials.class.getName());

    public static void main(String[] args) {
        try (Scanner scan = new Scanner(System.in)) {
            calculateFactorials(scan);
        }
    }

    private static void calculateFactorials(Scanner scan) {
        String keepGoing = "y";
        logger.info("Do you want to calculate factorials? (y/n): ");
        String initialResponse = scan.next();

        if (initialResponse.equalsIgnoreCase("y")) {
            while (keepGoing.equalsIgnoreCase("y")) {
                logger.info("Enter an integer: ");
                int val = scan.nextInt();

                try {
                    logger.log(Level.INFO, "Factorial({0}) = {1}", new
Object[]{val, MathUtils.factorial(val)});
                } catch (IllegalArgumentException e) {
                    logger.log(Level.WARNING, "Error: {0}",
e.getMessage());
                    logger.log(Level.WARNING, "Error calculating factorial:
{0}", e.getMessage());
                }

                logger.info("Another factorial? (y/n) ");
                keepGoing = scan.next();
            }
        } else {
            logger.info("Okay, skipping factorials.");
        }
    }
}
```

FactorialTest

```java
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package ihsan.pertemuan10;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintStream;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class FactorialsTest {
    private final ByteArrayOutputStream outContent = new
ByteArrayOutputStream();
    private final PrintStream originalOut;
    private final InputStream originalIn;
    private static final Logger logger =
Logger.getLogger(Factorials.class.getName());
    private Level originalLevel;

    FactorialsTest() {
        this.originalOut = System.out;
        this.originalIn = System.in;
    }

    @BeforeEach
    public void setUpStreams() {
        System.setOut(new PrintStream(this.outContent));
        this.originalLevel = logger.getLevel();
        logger.setLevel(Level.OFF);
    }

    @AfterEach
    public void restoreStreams() {
        System.setOut(this.originalOut);
        System.setIn(this.originalIn);
        logger.setLevel(this.originalLevel);
    }

    @Test
    void testMainMethodWithYesResponse() {
        String input = "y\n5\nn";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        Factorials.main(new String[0]);
    }

    @Test
    void testMainMethodWithNoResponse() {
        String input = "n";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        Factorials.main(new String[0]);
```

```
    }

    @Test
    void testMainMethodWithMultipleFactorials() {
        String input = "y\n5\ny\n10\nn";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        Factorials.main(new String[0]);
    }

    @Test
    void testMainMethodWithInvalidInput() {
        String input = "y\n-1\nn";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        Factorials.main(new String[0]);
    }

    @Test
    void testMainMethodWithLargeInput() {
        String input = "y\n20\nn";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        Factorials.main(new String[0]);
    }
}
```

MathUtils

```
package ihsan.pertemuan10;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MathUtils {

    private static final Logger logger =
Logger.getLogger(MathUtils.class.getName());

    private MathUtils() {
        // Private constructor to prevent instantiation
        throw new IllegalStateException("Utility class");
    }

    public static int factorial(int n) throws IllegalArgumentException {
        if (n < 0) {
            logger.log(Level.WARNING, "Attempted to calculate factorial of
negative number: {0}", n);
            throw new IllegalArgumentException("Factorial is not defined
for negative numbers");
        }
        if (n > 16) {
            logger.log(Level.WARNING, "Attempted to calculate factorial of
too large number: {0}", n);
            throw new IllegalArgumentException("Value too large, will cause
overflow (max is 16)");
        }
```

```
        logger.log(Level.FINE, "Calculating factorial of: {0}", n);
        int fac = 1;
        for (int i = n; i > 0; i--) {
            fac *= i;
        }
        return fac;
    }
}
```

Mathutiltest

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package ihsan.pertemuan10;

import java.lang.reflect.Constructor;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

class MathUtilsTest {
    MathUtilsTest() {
    }

    @Test
    void testFactorialOfZero() {
        Assertions.assertEquals(1, MathUtils.factorial(0));
    }

    @Test
    void testFactorialOfOne() {
        Assertions.assertEquals(1, MathUtils.factorial(1));
    }

    @ParameterizedTest
    @ValueSource(
        ints = {2, 3, 5, 10, 16}
    )
    void testFactorialOfPositiveNumbers(int n) {
        int result = MathUtils.factorial(n);
        Assertions.assertTrue(result > 0);
    }

    @Test
    void testFactorialOfFive() {
        Assertions.assertEquals(120, MathUtils.factorial(5));
    }

    @Test
    void testFactorialOfNegativeNumberThrowsException() {
        Exception exception =
(Exception)Assertions.assertThrows(IllegalArgumentException.class, () ->
MathUtils.factorial(-1));
        Assertions.assertEquals("Factorial is not defined for negative
numbers", exception.getMessage());
    }
```

```
    @Test
    void testFactorialOfLargeNumberThrowsException() {
        Exception exception =
(Exception)Assertions.assertThrows(IllegalArgumentException.class, () ->
MathUtils.factorial(17));
        Assertions.assertTrue(exception.getMessage().contains("Value too
large"));
    }

    @Test
    void testConstructorThrowsException() {
        try {
            Constructor<MathUtils> constructor =
MathUtils.class.getDeclaredConstructor();
            constructor.setAccessible(true);
            constructor.newInstance();
            Assertions.fail("Should have thrown an exception");
        } catch (Exception e) {
            Throwable cause = e.getCause();
            Assertions.assertNotNull(cause);
            Assertions.assertEquals(IllegalStateException.class,
cause.getClass());
            Assertions.assertEquals("Utility class", cause.getMessage());
        }

    }
}
```

Parseints

```
package ihsan.pertemuan10;
//  ParseInts.java
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ParseInts {

    private static final Logger logger =
Logger.getLogger(ParseInts.class.getName());

    public static void main(String[] args) {
        try (Scanner scan = new Scanner(System.in)) {
            parseAndSumIntegers(scan);
        }
    }

    private static void parseAndSumIntegers(Scanner scan) {
        int sum = 0;
        logger.info("Enter a line of text with numbers:");
        String line = scan.nextLine();

        try (Scanner scanLine = new Scanner(line)) {
            while (scanLine.hasNext()) {
                try {
                    sum += scanLine.nextInt();
                } catch (InputMismatchException e) {
                    logger.log(Level.WARNING, "Invalid input: {0}",
```

```
scanLine.next());
                }
            }
        }
        logger.log(Level.INFO, "The sum of the integers on this line is
{0}", sum);
    }
}
```

parseintstest

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package ihsan.pertemuan10;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintStream;
import java.lang.reflect.Method;
import java.util.NoSuchElementException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class ParseIntsTest {
    private final ByteArrayOutputStream outContent = new
ByteArrayOutputStream();
    private final PrintStream originalOut;
    private final InputStream originalIn;
    private static final Logger logger =
Logger.getLogger(ParseInts.class.getName());
    private Level originalLevel;

    ParseIntsTest() {
        this.originalOut = System.out;
        this.originalIn = System.in;
    }

    @BeforeEach
    public void setUpStreams() {
        System.setOut(new PrintStream(this.outContent));
        this.originalLevel = logger.getLevel();
        logger.setLevel(Level.OFF);
    }

    @AfterEach
    public void restoreStreams() {
        System.setOut(this.originalOut);
        System.setIn(this.originalIn);
        logger.setLevel(this.originalLevel);
    }

    @Test
```

```
    void testMainMethod() {
        String input = "10 20 30 40";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        ParseInts.main(new String[0]);
    }

    @Test
    void testMainMethodWithMixedInput() {
        String input = "10 abc 20 def 30";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        ParseInts.main(new String[0]);
    }

    @Test
    void testMainMethodWithEmptyInput() {
        String input = "\n";
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(input.getBytes());
        System.setIn(inputStream);
        ParseInts.main(new String[0]);
    }

    @Test
    void testParseAndSumIntegersWithValidNumbers() {
        String line = "10 20 30 40";
        Scanner scanLine = new Scanner(line);

        try {
            Method method =
ParseInts.class.getDeclaredMethod("parseAndSumIntegers", Scanner.class);
            method.setAccessible(true);
            method.invoke((Object)null, scanLine);
        } catch (Exception e) {
            Assertions.fail("Should not throw exception: " +
e.getMessage());
        }

    }

    @Test
    void testParseAndSumIntegersWithInvalidInput() {
        String line = "10 abc 30 xyz";
        Scanner scanLine = new Scanner(line);

        try {
            Method method =
ParseInts.class.getDeclaredMethod("parseAndSumIntegers", Scanner.class);
            method.setAccessible(true);
            method.invoke((Object)null, scanLine);
        } catch (Exception e) {
            Assertions.fail("Should not throw exception: " +
e.getMessage());
        }

    }

    @Test
```

```
    void testParseAndSumIntegersWithEmptyInput() {
        try {
            String line = "\n";
            Scanner scanLine = new Scanner(line);
            Method method =
ParseInts.class.getDeclaredMethod("parseAndSumIntegers", Scanner.class);
            method.setAccessible(true);
            method.invoke((Object)null, scanLine);
        } catch (NoSuchElementException var4) {
        } catch (Exception e) {
            if (!(e.getCause() instanceof NoSuchElementException)) {
                Assertions.fail("Unexpected exception: " + e.getMessage());
            }
        }

    }
}
```