Exercice 1 : Couverture de Code

1. Réécrire le code pour qu'il passe les vérifications de checkstyle, spotbugs et PMD.

   Réécriture de code

```
package info.net;
public final class Calcul {
  /**
  *Constructeur de la class.
  */
  private Calcul() {

  }
/**
*Calcul la somme de deux nombres .
*@param a
*@param b
*@return la somme de a et b
**/
  public static int somme(final int a, final int b) {
    return a + b;
  }
/**
*Renvoie la note, bornée par les valeurs min et max, exemples:.
* noteBornee(12.5,0.0,20.0) doit renvoyer 12.2
* noteBornee(25.0,0.0,20.0) doit renvoyer 20.0
* noteBornee(-2.0,0.0,20.0) doit renvoyer 0.0
*@param note
*@param min
*@param max
*@return resultat
**/
  public static double noteBornee(final double note,
                          final double min, final double max) {
  double resultat = note;
    if (note >= max) {
      resultat = max;
    }
    if (note <= min) {
      resultat = min;
    }
  return resultat;
  }
/**
*calcul la division.
*@param a
*@param b
*@return a / b si b != 0
```

```
* @throw IllegalArgumentException si b == 0
*/
   public static int division(final int a, final int b) {
      if (b == 0) {
         throw new IllegalArgumentException("b ne doit pas etre 0");
      }
   return a / b;
   }
}
```

2. Donner le rapport de couverture de code des tests unitaires de ‘CalculTest’ sur la classe ‘Calcul’.

   Voir repertoire ‘Rapport couverture de code CalculTest sur Calcul’

3. Expliquer pourquoi la couverture n’est pas complète (en couverture de ligne et en couverture de branche).

   La couverture n’est pas complète parce que le test unitaire pour la division ne teste pas le cas où on divise par zéro et que l’exception est levé, il n’y as pas de tests pour la méthode noteBornee qui devrait prendre en compte au moins 3 cas possible.

4. Proposer une amélioration des tests pour obtenir une couverture complète (donner le résultat de couverture obtenu).

   Amélioration du code

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.fail;
import org.junit.jupiter.api.Test;

public class CalculTest {
@Test
public void testConstructeur() { new Calcul(); }
@Test
public void testSomme() { assertEquals(5,Calcul.somme(2,3)); }
@Test
public void testDivision() { assertEquals(4,Calcul.division(8,2)); }
@Test
public void testDivision2(){

 try {
         Calcul.division(8,0);
          fail("cette ligne de code ne s'affiche pas");

 } catch(IllegalArgumentException e){
          System.out.println("exception attendue " + e.getMessage());
 }

}
```

```
@Test
public void testNoteBornee1(){

        assertEquals(12.5,Calcul.noteBornee(12.5,0.0,20.0));

}

@Test
public void testNoteBornee2(){

        assertEquals(20,Calcul.noteBornee(25.0,0.0,20.0));

}

@Test
public void testNoteBornee3(){

        assertEquals(0.0,Calcul.noteBornee(-12.5,0.0,20.0));

}

}
```

Exercice 2 : Compilation assistée

1. Organiser l'arborescence du code et des tests telle que prévue par l'outil 'maven'

Arborescence

```
/pom.xml
./src
./src/test
./src/test/java
./src/test/java/CalculTest.java
./src/main
./src/main/java
./src/main/java/Calcul.java
```

2. Construire un fichier "pom.xml" pour 'maven' permettant de
   - compiler le code
   - compiler les tests (en utilisant JUnit5)
   - exécuter les tests unitaires
   - générer les rapports de checkstyle, spotbugs et PMD
   - générer le rapport de couverture du code sur les tests unitaires (en utilisant JaCoCo)
   - générer la documentation javadoc
   - générer les différents rapports au format HTML

Contenue fichie pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <name>Activite2</name>
 <description>Devoir</description>
 <version>1.1</version>
 <groupId>default</groupId>
 <artifactId>Devoir2</artifactId>

 <dependencies><!-- dependendance du projet -->
  <dependency><!-- bibliotheque junit -->
```

```
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.4.2</version>
    <scope>test</scope>
   </dependency>
  </dependencies>

  <build>
   <plugins><!-- lister les plugins et leur version permet d'eviter que maven prenne celui qui
trouvera par defaut -->

    <plugin><!-- pour compiler -->
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.8.1</version>
     <configuration>
     <source>7</source>
     <target>7</target>
     </configuration>
    </plugin>

    <plugin>
          <artifactId>maven-project-info-reports-plugin</artifactId>
       <version>3.0.0</version>
    </plugin>

    <plugin><!-- pour l'execution des tests -->
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>3.0.0-M3</version>
          <configuration>
<testFailureIgnore>true</testFailureIgnore>
</configuration>
    </plugin>

    <plugin><!-- pour la generation de rapports -->
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-site-plugin</artifactId>
     <version>3.7.1</version>
     <configuration>
      <locales>fr,en</locales><!-- en francais par defaut -->
     </configuration>
    </plugin>
```

```
    <plugin><!-- pour la couverture de code avec JaCoco lors de l'exécution des tests JUnit -->
          <groupId>org.jacoco</groupId>
          <artifactId>jacoco-maven-plugin</artifactId>
          <version>0.8.4</version>
          <executions>
        <execution>
         <goals>
          <goal>prepare-agent</goal>
         </goals>
        </execution>
        <execution>
         <id>report</id>
         <phase>prepare-package</phase>
         <goals>
          <goal>report</goal>
         </goals>
        </execution>
          </executions>
    </plugin>


    <plugin><!-- pour l'analyse avec checkstyle -->
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-checkstyle-plugin</artifactId>
          <version>3.0.0</version>
    </plugin>

   </plugins>
  </build>

  <reporting><!-- on indique ici les rapports qu'on veut obtenir lors de la phase de generation de
rapports -->
   <plugins>
    <plugin>
     <groupId>org.jacoco</groupId>
     <artifactId>jacoco-maven-plugin</artifactId>
     <reportSets>
      <reportSet>
       <reports>
        <report>report</report>
       </reports>
      </reportSet>
     </reportSets>
    </plugin>
```

```
    <plugin><!-- pour avoir le rapport checkstyle -->
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-checkstyle-plugin</artifactId>
     <version>3.0.0</version>
     <reportSets>
      <reportSet>
       <reports>
        <report>checkstyle</report>
       </reports>
      </reportSet>
     </reportSets>
    </plugin>

    <plugin><!-- pour avoir le lien au code source dans les rapports -->
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-jxr-plugin</artifactId>
     <version>2.3</version>
    </plugin>

    <plugin><!-- pour la verification du code avec PMD -->
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-pmd-plugin</artifactId>
     <version>3.12.0</version>
    </plugin>

    <plugin><!-- pour avoir le rapport surfire (execution des tests) -->
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-surefire-report-plugin</artifactId>
     <version>3.0.0-M3</version>
    </plugin>

   </plugins>
  </reporting>

  <properties>
   <!-- encodage du code source -->
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

</project>
```

3. Donner la commande permettant d'exécuter toutes les lignes de la question 2 en une fois.

   Commande permettant d'executé toutes les lignes en une fois.
   mvn clean compile test site:site

4. Donner les rapports HTML générés.

   Les rapport HTML son situé dans le repertoire \target\site

5. Déposer les données utiles (pom.xml, src) dans le dossier "activite2" du dépôt (SVN ou GIT) utilisé précédemment.

   Dépôt github
   https://github.com/JeanGarino/Activit-s-C306/tree/main/activite2/exercice1

Exercice 3 : Sudoku

1. Ecrire les classes

HorsBornesException (exception levée lorsque les paramètres de positions sortent de la grille),
ValeurImpossibleException (exception levée lorsqu'on cherche à mettre une valeur dans la grille qui ne respecte pas les
règles),
CaractereInterditException (exception levée lorsqu'on cherche à mettre une valeur dans la grille qui ne fait pas partie du jeu
de valeurs utilisé par la grille)
ValeurInitialeModificationException (exception levée lorsqu'on cherche à modifier une valeur initiale de la grille)

HorsBornesException

```
public class HorsBornesException extends Exception{

public HorsBornesException(String message){

super(message);
}
}
```

ValeurImpossibleException

```
public class ValeurImpossibleException extends Exception{

public ValeurImpossibleException(String message){

super(message);
}
}
```

CaractereInterditException

```
public class CaractereInterditException extends Exception{

public CaractereInterditException(String message){
```

super(message);

}

}

ValeurInitialeModificationException

public class ValeurInitialeModificationException extends Exception{

public ValeurInitialeModificationException(String message){

super(message);

}

}