Sumar

Lista de Figuri

1. Setarea culorilor pentru vizualizarea acoperirii codului în IntelliJ IDEA

1. În meniul **File** ---> **Settings…** ---> **Editor** (dublu click) ---> **Color Scheme** ---> **General** ---> se deschide nodul **Line Coverage** din lista din partea dreaptă;
2. aici se vor seta culorile pentru cele 3 tipuri de explorare a codului sursă: **Full**, **Partial**, **Uncovered**;
3. în continuare sunt descrişi paşii pentru setarea culorii de **Background**, în locul celei de **Foreground**. Fiecare echipă poate decide nivelul (**Foreground** sau **Background**) pentru care doreşte să utilizeze codul de culori specificat (verde, galben, roşu).
4. se selectează **Full**, se debifează opţiunea **Foreground**; pentru **Background** se completează culoarea **CCFFCC** (verde), apoi **Choose** (vezi Figure 1);
5. similar:
   - pentru **Partial** se setează culoarea **Background** la **FFFFCC** (galben) şi se debifează opţiunea Foreground;
   - pentru **Uncovered**, se setează culoarea **Background** la **FFCCCC** (roşu) şi se debifează opţiunea **Foreground**.
6. **OK** pentru a salva setările referitoare la culoare.

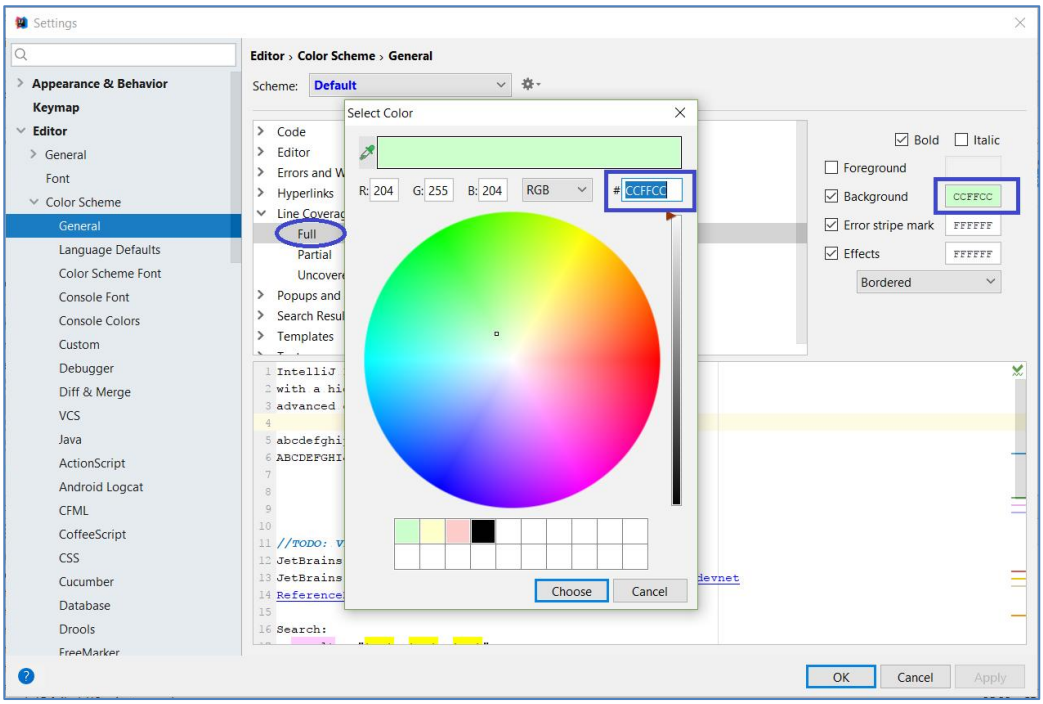


Figure 1. Configurarea culorilor pentru Line Coverage

2. Configurarea IntelliJ IDEA pentru rularea cu Coverage

1. Se alege clasa cu teste care va fi rulată;
2. În meniul **Run** ---> **Edit Configurations**… se creează o configuraţie de rulare de tip JUnit;
3. Se completează:
   - **Alegerea clasei de test** (vezi Figure 2):
     - **Test kind**: *Class*;
     - **Class**: numele clasei cu teste, e.g., *tasks.TaskTest*;
     - se alege opţiunea **Modify Options** pentru configurarea acoperii la execuţia testelor;

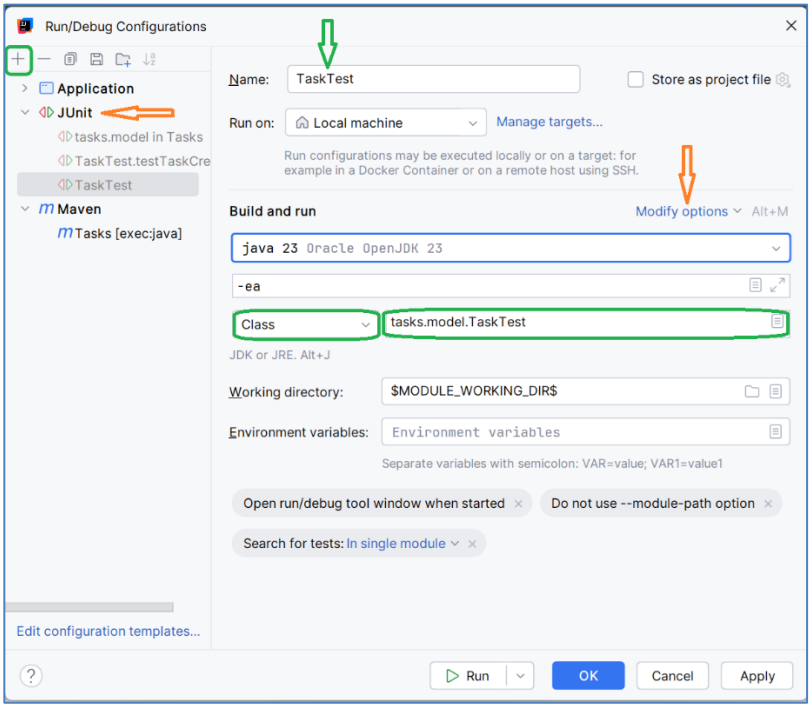


Figure 2 Algerea clasei cu teste

- **configurarea opţiunilor de acoperire** (vezi Figure 3, Figure 4):
  - **Specify classes and packages** – se aleg clasele/pachetele pentru care să se realizeze (înregistreze sau calculeze) acoperirea la testare, e.g., clasa ***Task***¸ pachetul ***repository***;

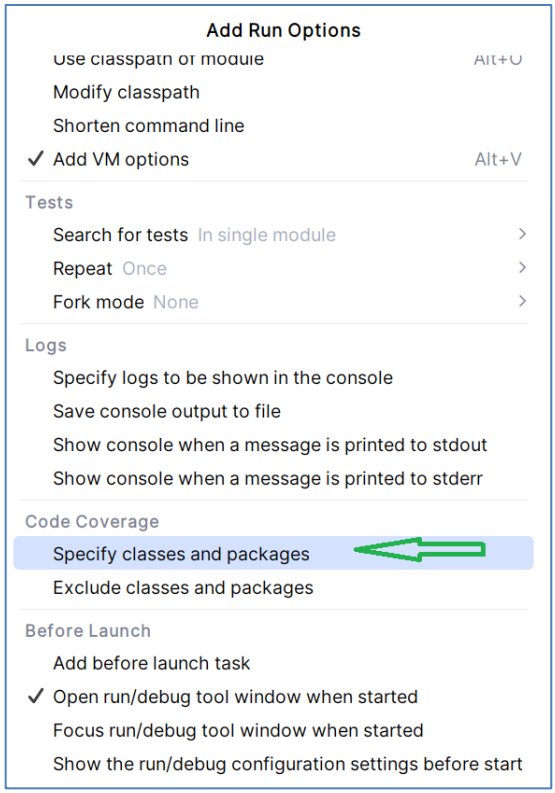


Figure 3 Configurarea opţiunilor de acoperire

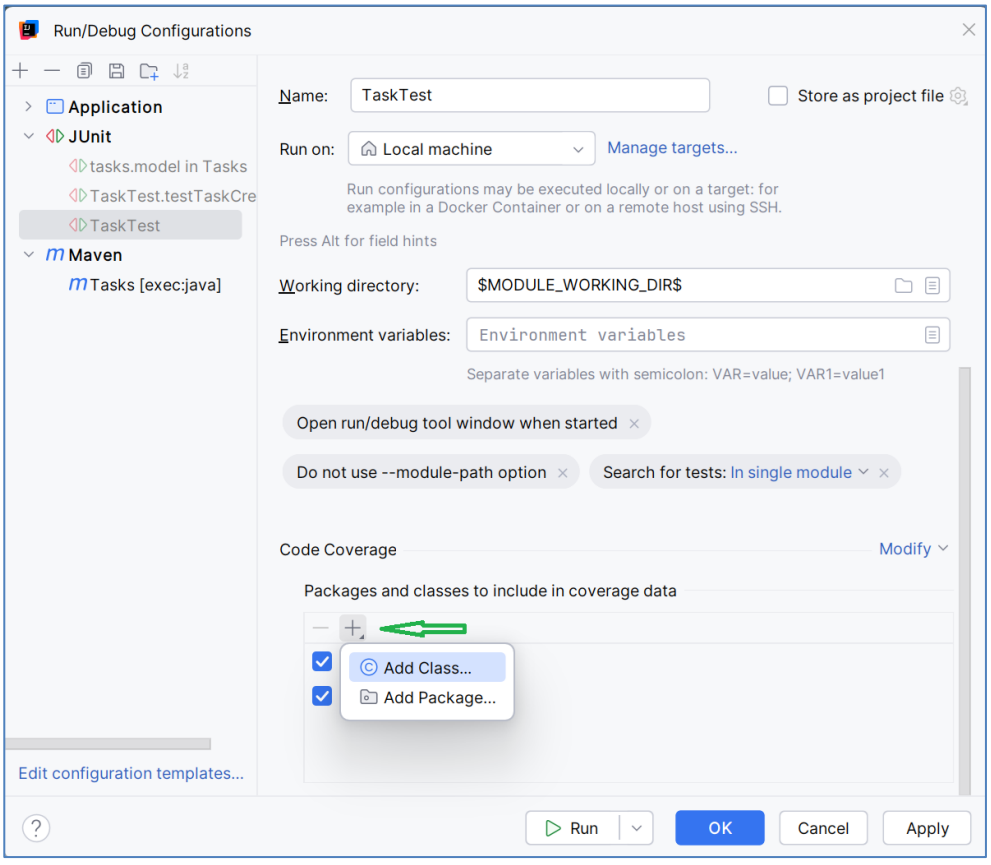


Figure 4. Configurarea opţiunilor de acoperire (forma finală)

3. Rularea testelor. Calculul acoperirii codului sursă testat

1. Meniul **Run** ---> **Run '*[numele configurației de rulare]*' with Coverage;**
2. După rulare se va deschide fereastra **Coverage** (vezi Figure 5);
3. Pentru fiecare pachet, prin dublu-click se poate vizualiza gradul de acoperire pentru clasă, metode, linii de cod și ramificații (vezi Figure 5, Figure 6);
4. Pentru a vizualiza acoperirea folosind codul de culori (verde, galben, roșu) la nivel de *background/foreground*, se realizează dublu-click în fereastra **Coverage** pe numele unei clase pentru care s-a efectuat rularea cu calculul/evaluarea acoperirii.

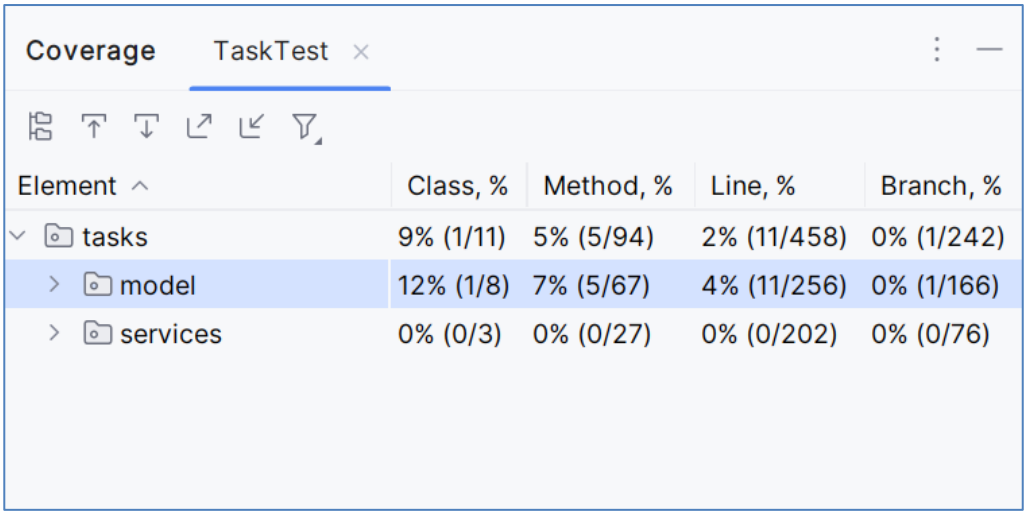


Figure 5. Fereastra Coverage – configurația de rulare *TaskTest* pentru pachetele selectate*l*

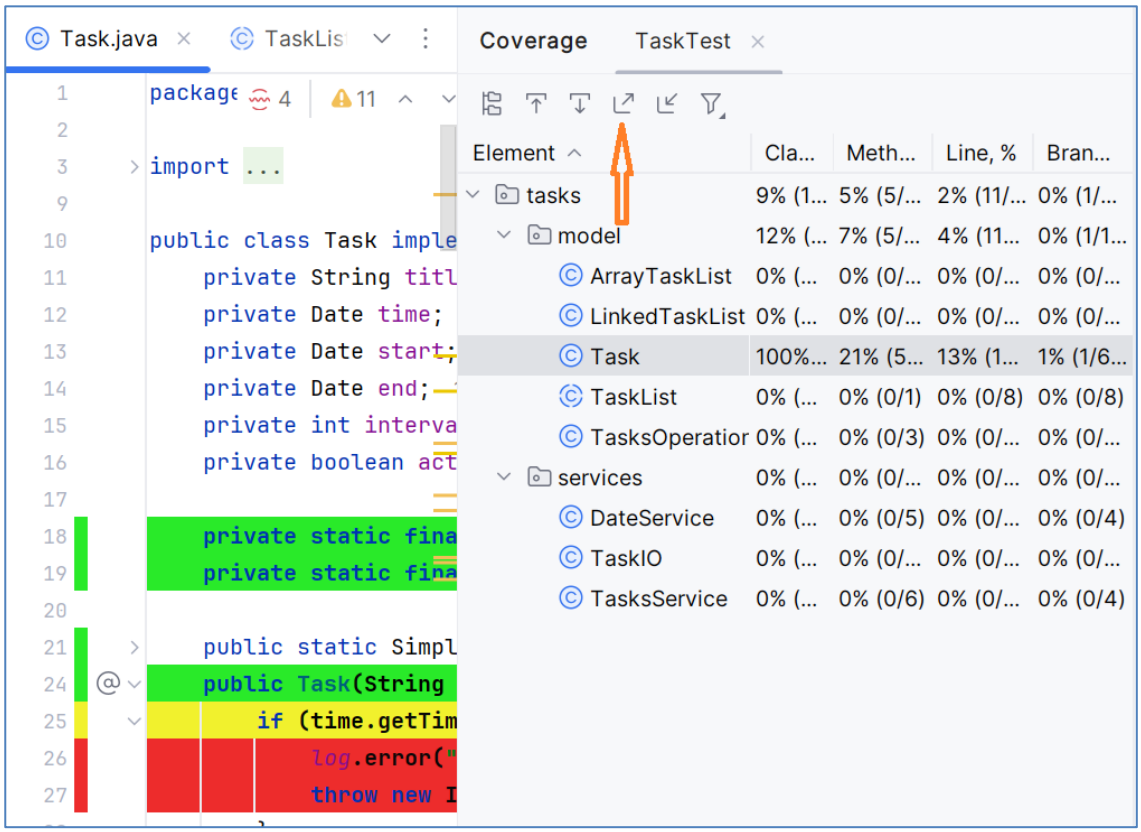


Figure 6 Fereastra *Coverage* – configurația de rulare *TaskTest* pentru clasa *Task* din pachetul *model*

4. Vizualizarea acoperirii codului sursă testat

1. Meniul **Run** ---> **Manage Coverage Reports** (vezi Figure 7);
2. Se alege o configurație de rulare care e bazată pe *IntelliJ IDEA* (vezi Figure 8), apoi se alege **Show selected**;
3. În **Project Explorer**, în dreptul fiecărei entități pentru care s-a monitorizat/calculat acoperirea, se afișează procentul de acoperire (vezi Figure 9);
4. După execuția testelor, în frame-ul de editare a codului sursă, liniile de cod sursă sunt colorate corespunzător nivelului de acoperire (verde-*full*, galben-*parțial*, roșu-*neacoperit*) (vezi Figure 9).

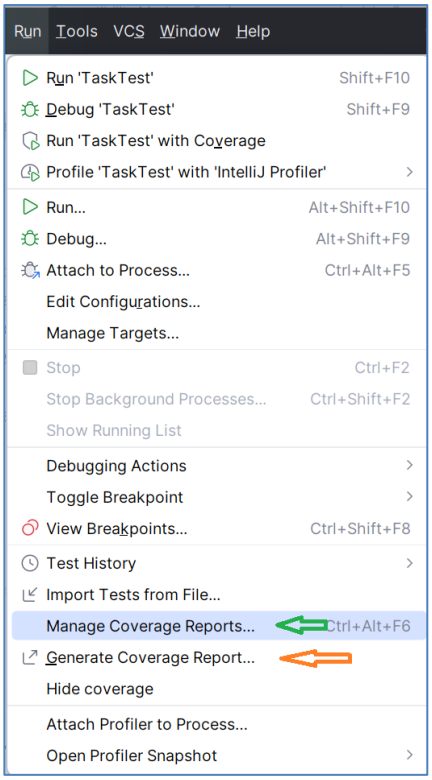


Figure 7. Alegerea opțiunii pentru vizualizarea acoperirii cu teste a codului rulat

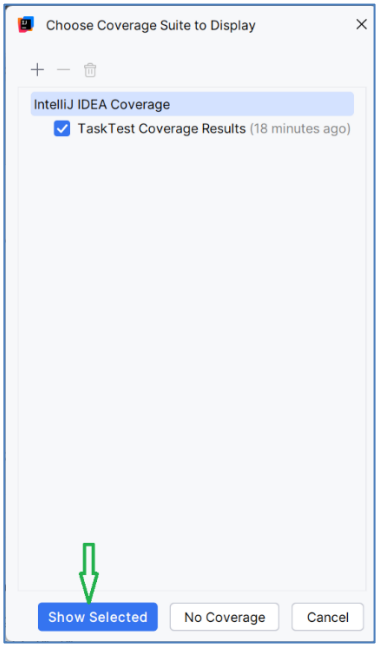


Figure 8. Alegerea configurației de rulare care folosește IntelliJ IDEA

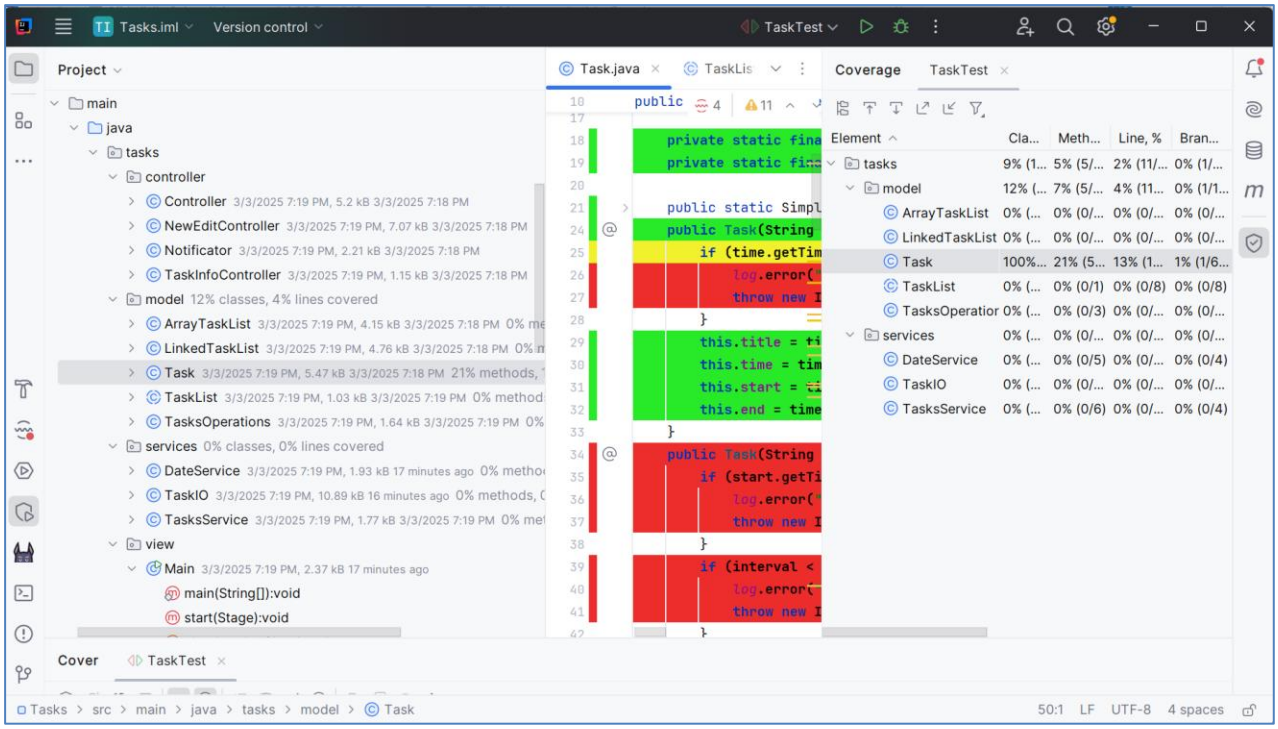


Figure 9. Vizualizarea acoperirii în procente şi culori pentru configuraţia de rulare aleasă

5. Generarea şi vizualizarea raportului de acoperire

Accesarea opţiunii de generare a raportului de acoperire se poate face în două moduri:

Varianta 1:

- Meniul **Run** ---> **Generate Coverage Report** (vezi Figure 7);

Varianta 2:

- Fereastra **Coverage, butonul marcat cu verde (Generate Coverage Report)** (vezi Figure 6).

Apoi:

1. Se alege folderul în care se salvează raportul;
2. Se bifează opţiunea **Open generate HTML in browser**, apoi **Save** (vezi Figure 10);
3. Se va deschide implicit raportul de acoperire într-un browser web (vezi Figure 11);
4. Pentru fiecare clasă, prin dublu-click se poate observa (**vizual**, prin culorile verde, galben, roşu), cât şi **în rezumatul raportului** (prin valoarea procentului din tabel) gradul de acoperire cu teste pentru clase, metode şi liniile de cod sursă (vezi Figure 12).

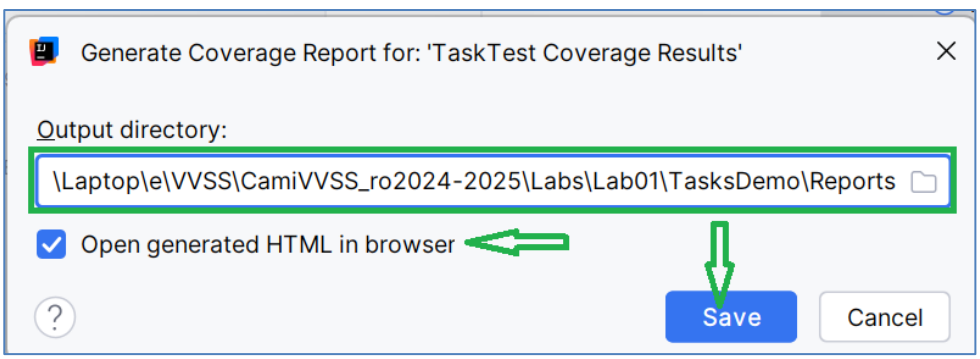
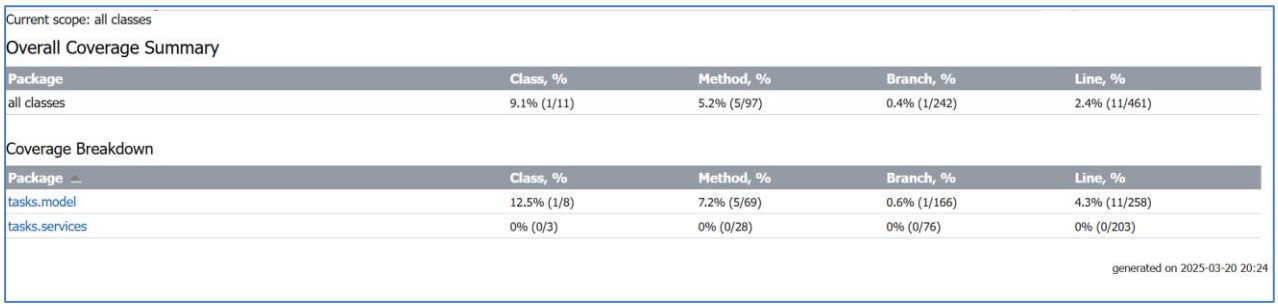


Figure 10. Configurarea opţiunii de generare a raportului de acoperire (deschidere in browser, folder pentru salvare)



Current scope: all classes

Overall Coverage Summary

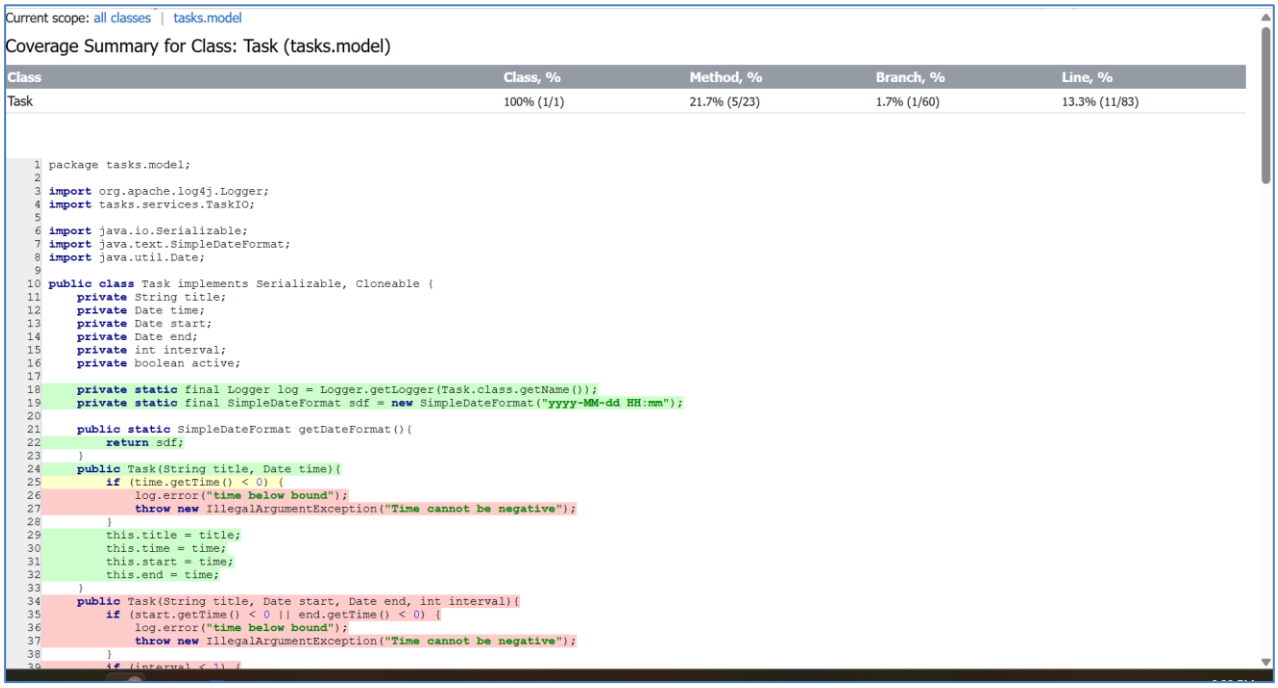
Package	Class, %	Method, %	Branch, %	Line, %
all classes	9.1% (1/11)	5.2% (5/97)	0.4% (1/242)	2.4% (11/461)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
tasks.model	12.5% (1/8)	7.2% (5/69)	0.6% (1/166)	4.3% (11/258)
tasks.services	0% (0/3)	0% (0/28)	0% (0/76)	0% (0/203)

generated on 2025-03-20 20:24

Figure 11. Raportul de acoperire cu teste pentru clasele din pachetele selectate după de rularea testelor



Current scope: all classes | tasks.model

Coverage Summary for Class: Task (tasks.model)

Class	Class, %	Method, %	Branch, %	Line, %
Task	100% (1/1)	21.7% (5/23)	1.7% (1/60)	13.3% (11/83)

```
1  package tasks.model;
2
3  import org.apache.log4j.Logger;
4  import tasks.services.TaskIO;
5
6  import java.io.Serializable;
7  import java.text.SimpleDateFormat;
8  import java.util.Date;
9
10 public class Task implements Serializable, Cloneable {
11     private String title;
12     private Date time;
13     private Date start;
14     private Date end;
15     private int interval;
16     private boolean active;
17
18     private static final Logger log = Logger.getLogger(Task.class.getName());
19     private static final SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");
20
21     public static SimpleDateFormat getDateFormat(){
22         return sdf;
23     }
24     public Task(String title, Date time){
25         if (time.getTime() < 0) {
26             log.error("time below bound");
27             throw new IllegalArgumentException("Time cannot be negative");
28         }
29         this.title = title;
30         this.time = time;
31         this.start = time;
32         this.end = time;
33     }
34     public Task(String title, Date start, Date end, int interval){
35         if (start.getTime() < 0 || end.getTime() < 0) {
36             log.error("time below bound");
37             throw new IllegalArgumentException("Time cannot be negative");
38         }
39         if (interval < 1) {
```

Figure 12. Raportul de acoperire cu teste pentru clasa `Task`