



En introduktion till att skriva tester i Java

Testning

Innehåll

01 Introduktion - vad är testning?

02 JUnit, filstruktur och ett exempel

03 Termer inom testning och JUnit

04 Introduktion till unit testing

05 Mocking

06 Introduktion till integration testing

07 Testdriven utveckling med ett exempel



Vad är testning?

01

Introduktion

Vad är testning?

Testning innebär att skriva kod som testar att annan kod fungerar.

Testning är ett sätt att hitta fel i kod, säkerställa att alla krav finns med och förbättra system.

Det finns olika typer av testning som unit(-testing), integration, user, performance och automated.

De flesta språk har specialiserade ramverk för testning.

- C# - XUnit, Moq
- Go - Ginkgo, httpexpect
- Java - JUnit, Mockito

Varför testning?

Testning kan göras manuellt, men det tar tid. Kod-tester är till för att kunna köras ofta och snabbt. Människor gör också misstag och glömmer saker, vilket inte stämmer för kod.

Testning är också ett sätt att beskriva krav. Ifall ett test misslyckas saknas något eller flera krav.

Kod blir ofta mer flexibel på grund av tester, då testning endast kan göras på kod med flexibel struktur.

Pitfalls och blind spots med testning

- Testning löser inte alla buggar
- Testning hittar inte alla buggar
- Testning kan skapa en falsk trygghetskänsla
- Testning tar mer tid — ibland mycket mer
- Testning kan leda till komplexitet och övertänkande
- Testkod är också kod som måste underhållas och dokumenteras



Ett populärt Java testramverk

02

JUnit

Vad är JUnit?

JUnit är ett testramverk för Java med stöd för unit testing och integration testing främst.

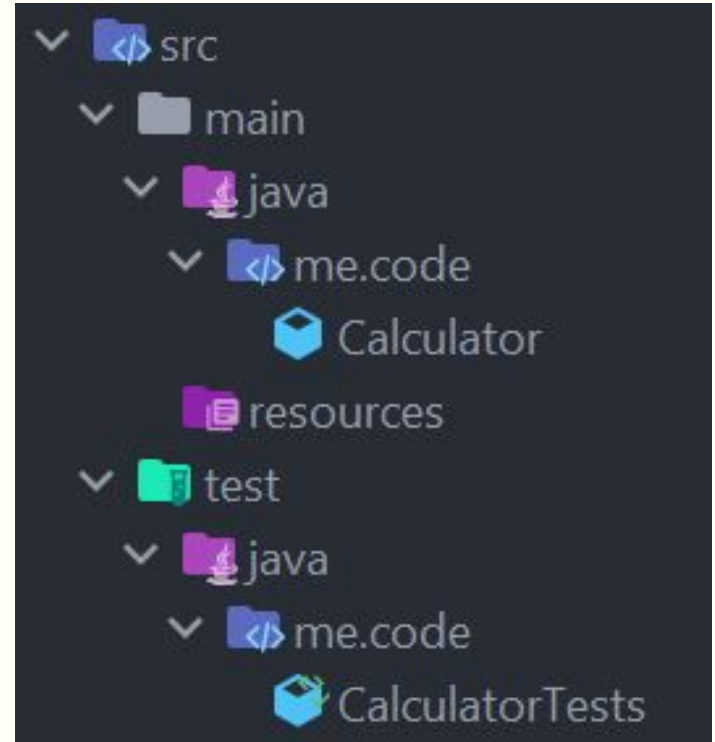
- Metoder för tester
- Klasser för organisering
- Annotationer för data och dokumentation
- Assertions för villkor
- Inbyggt stöd i IntelliJ



Filstruktur med JUnit

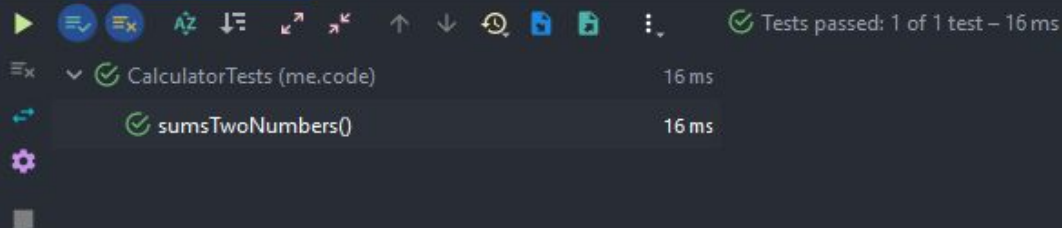
Kod delas upp i två delar: original kod och testkod

- Java packages finns i båda mappar
- Testfiler heter ofta samma som original kod med "Tests" på slutet (Service -> ServiceTests)
- Tester kan delas upp i unit tests och integration tests med olika filer



Test exempel med JUnit

```
public class CalculatorTests {  
  
    @Test  
    void sumsTwoNumbers() {  
        var calculator = new Calculator();  
        var result :int = calculator.add( a: 5, b: 3);  
        Assertions.assertEquals( expected: 8, result);  
    }  
}
```





03

Termer

Begrepp och termer

Test

En funktion som testar en specifik del av kod.

Ett test består oftast av flera "assertions" och testar typiskt sett en "unit".

Unit

En del kod som kan testas, ofta en funktion eller klass.

En "unit" har ingen fast definition, men typiskt sätt så är det bättre med små enheter. Det är bättre att testa en individuell funktion än en hel klass.

Assertion

En funktion som gör själva testet.

- Equals
- Contains
- Null, Not equals
- Is true, Is false

Begrepp och termer

Red & green

Ett sätt att beskriva ett test som har exekverats.

Red innebär att testet misslyckades och måste lösas.

Green innebär att testet lyckades och man kan gå vidare.

Refactor

Att ändra och förbättra kod som testas.

Refactoring sker oftast efter att ett test har misslyckats, eller efter det lyckas men man vet att förbättringar kan göras.

Lifecycles

Funktioner som körs innan, mellan och efter tester.

De kan användas för att skapa databaskopplingar, skapa filer, radera filer, "städa upp" och mer.

Begrepp och termer

Given When Then

Ett sätt att strukturera tester på.

Given – all data läggs här.

When – kör kod som skall testas.

Then – kör assertions och tester.

Arrange Act Assert

Ett annat namn för "given when then".

Arrange = Given

Act = When

Assert = Then

Mocking

Ett sätt att isolera komponenter för bättre enhetstestning.

JUnit annotations

@Test

@RepeatedTest

@ParameterizedTest

@Disabled

Applieras på funktioner för att definiera dem som tester.

@Test – vanligt test.

@RepeatedTest – upprepande test.

@ParameterizedTest – test som inkluderar data.

@Disabled – stäng av test

@DisplayName

Ange ett beskrivande namn på ett test. Detta är bra för dokumentation och tydlighet.

@Timeout

Ange en tid och testet ser till att köras inom tiden. Om inte så misslyckas testet.

Detta är bra för applikationer som har höga krav på prestanda.



Tester som involverar små enheter

04

Unit testing

Vad är unit testning?

Unit testing innebär att testa mindre delar kod, så kallade enheter (units på engelska.)

En enhet är ofta en individuell funktion eller ibland en liten klass

Syften och fördelar med enhetstester inkluderar:

- Tidig bughantering
- Verifiera att komponenter fungerar i isolation
- Säkerställ att krav är med
- Dokumentera kod
- Förbättra design och struktur av kod
- Motivera och skapa en känsla av säkerhet

Enheter och beroenden

Kod har ofta beroenden, som databaskopplingar, inställningar, referenser till filer och mer. Dessa gör testning svårt.

En enhet skall ha så få beroenden som möjligt, helst inga. Om ett beroende finns skall det kunna styras (vilket kan göras med mocking exempelvis.)

Kod som har många ansvar — eller beroenden — brukar inte anses som en enhet.

Funktioner som refererar till global information är också svåra att testa. Därför är det bra att funktioner tar in parametrar för all information de behöver.



Ett sätt att isolera enheter

05

Mocking

Enheter och beroenden, igen

Kod måste ibland ha dessa beroenden. Om ett program skall använda en databas så måste det finnas en koppling till databasen någonstans.

Beroenden kan "styras" med hjälp av mocking — ett sätt att skapa låtsas objekt som simulerar verkligheten.

I Java kan ramverket "Mockito" användas för mocking.



Tester som involverar flera enheter och system

06

Integration testing

Vad är integration testing?

Integration testing innebär att testa flera delar kod, så att de fungerar tillsammans. Man kan säga att integration testing är som att ta flera "enheter" och testa dem tillsammans.

Syften och fördelar med integrationstester inkluderar:

- Verklighetsbaserad testning
- Verifiera att komponenter fungerar tillsammans
- Testa mycket samtidigt (effektivt)
- Se till att hela systemet fungerar

Typer av integrationstester

01. Top-down

Börja med att testa alla komponenter längst upp i logiskt hierarki.

Main → Menu →

Command → Service

02. Bottom-up

Börja med att testa alla komponenter längst ned i logiskt hierarki.

Service → Command →

Menu → Main

03. Hybrid

Blanda tester av olika nivåer.

Main → Menu

Command → Service

Data → Service

04. Big-bang

Testa den första nivån, eller första komponenten, bara.

Main → Alla



Skriv tester först, och sedan koden

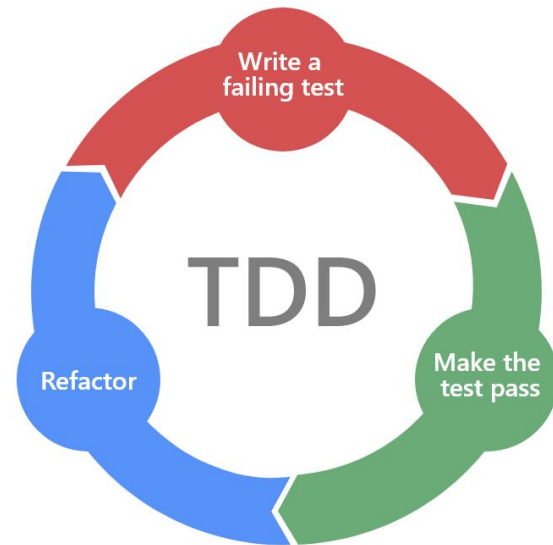
07

Testdriven utveckling

Testdriven utveckling

Ett sätt att skriva kod som innebär att tester skapas först, och koden följer.

- Kräver mer planering och mer tydlig struktur
- Färre buggar och problem
- Automatiskt bättre APIer
- Bättre val av design
- Leder ofta till mer abstrakt och flexibel kod
- Leder ofta till långsam kod på grund av abstraktioner
- Tar mer tid på grund av planering och skrivning av tester



Vad är TDD?

- Står för 'Test Driven Development'
- En metodik där tester skrivs innan programkoden
- 5 steg: skriv test, kör test, lös test, kör test & förbättra kod
- 2 extra steg: kör test efter förbättring och gå till nästa test

Filstruktur

```
src
├── main
│   ├── java
│   │   ├── me.code
│   │   │   ├── controllers
│   │   │   │   └── TodoController
│   │   │   ├── data
│   │   │   │   └── Todo
│   │   │   ├── exceptions
│   │   │   │   ├── TodoAlreadyExistsException
│   │   │   │   ├── TodoDoesNotExistException
│   │   │   │   └── TodoMustHaveNameException
│   │   │   ├── repositories
│   │   │   │   ├── InMemoryTodoRepository
│   │   │   │   └── TodoRepository
│   │   │   ├── services
│   │   │   │   └── TodoService
│   │   │   └── App
│   │   └── resources
│   └── test
│       ├── java
│       │   ├── me.code
│       │   │   ├── InMemoryTodoRepositoryTests
│       │   │   ├── TodoControllerTests
│       │   │   └── TodoServiceTests
```

Specifikationer

Repository

- Spara todos
 - Om ett null värde sparas skall 'NullPointerException' kastas
- Radera todos
 - Om namnet är null skall 'NullPointerException' kastas
 - Om en todo inte finns skall 'TodoDoesNotExist' kastas

Service

- Skapa todos
 - Om en todo skapas dubbelt skall 'TodoAlreadyExistsException' kastas
 - Om en todo skapas utan namn skall 'TodoMustHaveNameException' kastas
- Radera todos
 - Om en todo inte finns skall 'TodoDoesNotExistException' kastas
- Kopplas till TodoRepository

Steg 1 – Skriv test

InMemoryTodoRepositoryTests

```
public class InMemoryTodoRepositoryTests {  
  
    TodoRepository repository;  
  
    @BeforeEach  
    void setup() {  
        repository = new InMemoryTodoRepository();  
    }  
  
    @Test  
    @DisplayName("Save todo to storage correctly")  
    void save_stores_todo() {  
        // given  
        var todo = new Todo( name: "Städa", description: "Städa mitt rum kl 15:00.");  
  
        // when/then  
        Assertions.assertDoesNotThrow(() → repository.save(todo));  
  
        // then  
        Assertions.assertEquals( expected: 1, repository.count());  
    }  
}
```

Steg 2 – Kör testet InMemoryTodoRepositoryTests



The screenshot shows an IDE's test runner interface. At the top, a status bar indicates "Tests failed: 1 of 1 test - 15 ms". Below this, a tree view shows the test suite "InMemoryTodoRepositoryTests (me.code)" with a duration of "15 ms". Underneath, a single test "Save todo to storage correctly" is listed with a duration of "15 ms". To the right of the test name, the failure details are displayed: "org.opentest4j.AssertionFailedError:", "Expected :1", "Actual :0", and a link "<Click to see difference>". The background of the IDE window is dark, and the text is light gray.

```
Tests failed: 1 of 1 test - 15 ms  
InMemoryTodoRepositoryTests (me.code) 15 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...  
  Save todo to storage correctly 15 ms  
    org.opentest4j.AssertionFailedError:  
    Expected :1  
    Actual   :0  
    <Click to see difference>
```

Steg 3 – Lös testet

InMemoryTodoRepository

```
public class InMemoryTodoRepository implements TodoRepository {

    private final Map<String, Todo> data = new HashMap<>();

    @Override
    public void save(Todo todo) {
        data.put(todo.getName(), todo);
    }

    @Override
    public int count() {
        return data.size();
    }

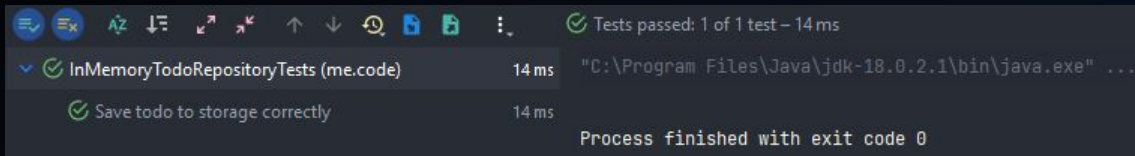
    @Override
    public void delete(String name) {

    }

}
```

Steg 4 – Kör testet igen

InMemoryTodoRepositoryTests



The screenshot shows an IDE's test runner interface. At the top, a status bar indicates "Tests passed: 1 of 1 test - 14 ms". Below this, a list of test results is displayed. The first entry is "InMemoryTodoRepositoryTests (me.code)" with a duration of "14 ms" and a command line snippet: "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ... The second entry is "Save todo to storage correctly" with a duration of "14 ms". At the bottom, a message states "Process finished with exit code 0".

Test Name	Duration	Command Line
InMemoryTodoRepositoryTests (me.code)	14 ms	"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
Save todo to storage correctly	14 ms	

Process finished with exit code 0

Steg 5 & 6 – Förbättra koden InMemoryTodoRepository

```
public class InMemoryTodoRepository implements TodoRepository {  
  
    private final Map<String, Todo> data = new HashMap<>();  
  
    @Override  
    public void save(Todo todo) {  
        data.put(todo.getName(), todo);  
    }  
  
    @Override  
    public int count() {  
        return data.size();  
    }  
  
    @Override  
    public void delete(String name) {  
  
    }  
}
```

Steg 7 – Repetera för nästa test

- Spara todos ✓
 - Om ett null värde sparas skall 'NullPointerException' kastas
- Radera todos
 - Om namnet är null skall 'NullPointerException' kastas
 - Om en todo inte finns skall 'TodoDoesNotExist' kastas

Steg 1 – Skriv test

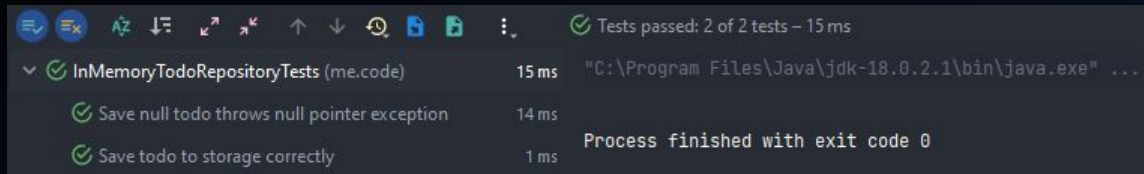
InMemoryTodoRepositoryTests

```
@Test
@DisplayName("Save null todo throws null pointer exception")
void save_null_todo_throws_null_pointer() {
    // when/then
    Assertions.assertThrows(
        NullPointerException.class,
        () → repository.save( todo: null)
    );

    // then
    Assertions.assertEquals( expected: 0, repository.count());
}
```

Steg 2 – Kör testet

InMemoryTodoRepositoryTests



Steg 3 & 4 – Testet redan löst InMemoryTodoRepository

```
public class InMemoryTodoRepository implements TodoRepository {  
  
    private final Map<String, Todo> data = new HashMap<>();  
  
    @Override  
    public void save(Todo todo) {  
        data.put(todo.getName(), todo);  
    }  
  
    @Override  
    public int count() {  
        return data.size();  
    }  
  
    @Override  
    public void delete(String name) {  
  
    }  
}
```

Steg 5 – Förbättra koden

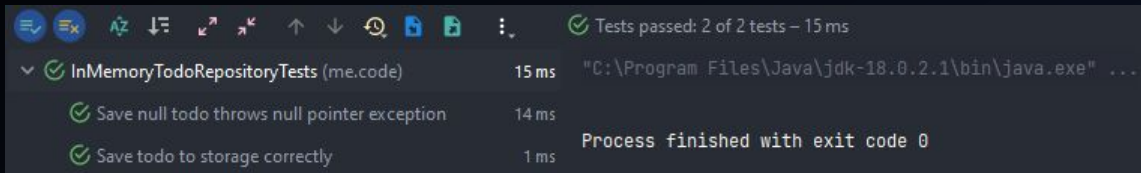
InMemoryTodoRepository

```
@Override
public void save(Todo todo) {
    if (todo == null) {
        throw new NullPointerException("cannot save null value");
    }

    data.put(todo.getName(), todo);
}
```

Steg 6 – Kör testet igen

InMemoryTodoRepositoryTests



Steg 7 – Repetera för nästa test

- Spara todos ✓
 - Om ett null värde sparas skall 'NullPointerException' kastas ✓
- Radera todos
 - Om namnet är null skall 'NullPointerException' kastas
 - Om en todo inte finns skall 'TodoDoesNotExist' kastas

Steg 1 – Skriv test

InMemoryTodoRepositoryTests

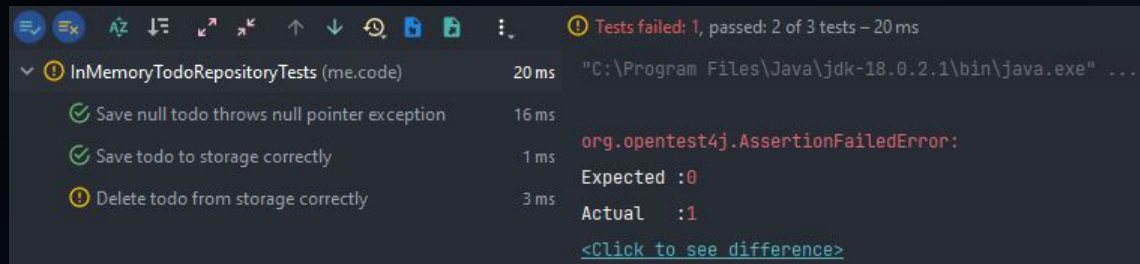
```
@Test
@DisplayName("Delete todo from storage correctly")
void delete_removes_todo() {
    // given
    var name = "Städa";
    var todo = new Todo(name, description: "Städa mitt rum kl 15:00.");
    repository.save(todo);

    // when/then
    Assertions.assertDoesNotThrow(() → repository.delete(name));

    // then
    Assertions.assertEquals( expected: 0, repository.count());
}
```

Steg 2 – Kör testet

InMemoryTodoRepositoryTests



```
Tests failed: 1, passed: 2 of 3 tests – 20 ms

InMemoryTodoRepositoryTests (me.code) 20 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
  ✓ Save null todo throws null pointer exception 16 ms
  ✓ Save todo to storage correctly 1 ms
  ✗ Delete todo from storage correctly 3 ms
    org.opentest4j.AssertionFailedError:
      Expected :0
      Actual   :1
      <Click to see difference>
```

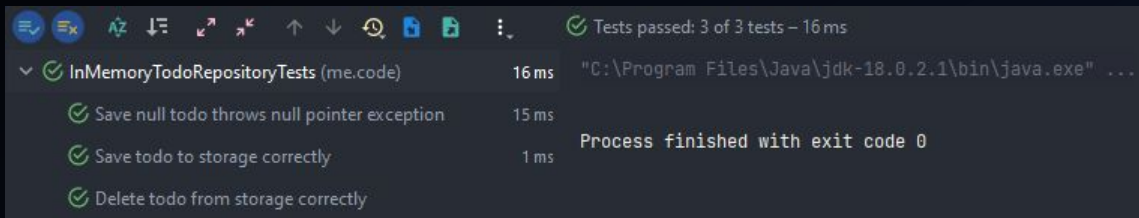
Steg 3 – Lös testet

```
@Override  
public void delete(String name) {  
    data.remove(name);  
}
```

InMemoryTodoRepository

Steg 4 – Kör testet igen

InMemoryTodoRepositoryTests



The screenshot shows a test runner interface with a toolbar at the top containing icons for test execution, search, and other functions. The main area displays the test results for `InMemoryTodoRepositoryTests`. The tests passed, and the process finished with exit code 0.

Test Name	Duration	Status
<code>InMemoryTodoRepositoryTests</code> (me.code)	16 ms	Passed
Save null todo throws null pointer exception	15 ms	Passed
Save todo to storage correctly	1 ms	Passed
Delete todo from storage correctly		Passed

Tests passed: 3 of 3 tests – 16 ms

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...

Process finished with exit code 0

Steg 5 & 6 – Förbättra koden

InMemoryTodoRepository

```
public class InMemoryTodoRepository implements TodoRepository {

    private final Map<String, Todo> data = new HashMap<>();

    @Override
    public void save(Todo todo) {
        if (todo == null) {
            throw new NullPointerException("cannot save null value");
        }

        data.put(todo.getName(), todo);
    }

    @Override
    public int count() {
        return data.size();
    }

    @Override
    public void delete(String name) {
        data.remove(name);
    }
}
```

Steg 7 – Repetera för nästa test

- Spara todos ✓
 - Om ett null värde sparas skall 'NullPointerException' kastas ✓
- Radera todos ✓
 - Om namnet är null skall 'NullPointerException' kastas
 - Om en todo inte finns skall 'TodoDoesNotExist' kastas

Steg 1 – Skriv test

InMemoryTodoRepositoryTests

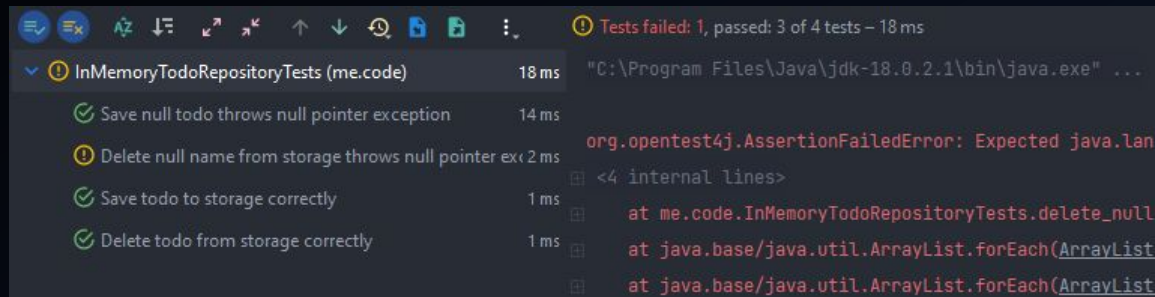
```
@Test
@DisplayName("Delete null name from storage throws null pointer exception")
void delete_null_todo_throws_null_pointer() {
    // given
    var todo = new Todo( name: "Städa", description: "Städa mitt rum kl 15:00.");
    repository.save(todo);

    // when/then
    Assertions.assertThrows(
        NullPointerException.class,
        () → repository.delete( name: null)
    );

    // then
    Assertions.assertEquals( expected: 1, repository.count());
}
```

Steg 2 – Kör testet

InMemoryTodoRepositoryTests



```
Tests failed: 1, passed: 3 of 4 tests - 18 ms

InMemoryTodoRepositoryTests (me.code) 18 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...

  ✓ Save null todo throws null pointer exception 14 ms
  ✗ Delete null name from storage throws null pointer exception 2 ms
    org.opentest4j.AssertionFailedError: Expected java.lang.NullPointerException but got java.lang.AssertionError
    <4 internal lines>
    at me.code.InMemoryTodoRepositoryTests.delete_null
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
  ✓ Save todo to storage correctly 1 ms
  ✓ Delete todo from storage correctly 1 ms
```


Steg 3 – Lös testet

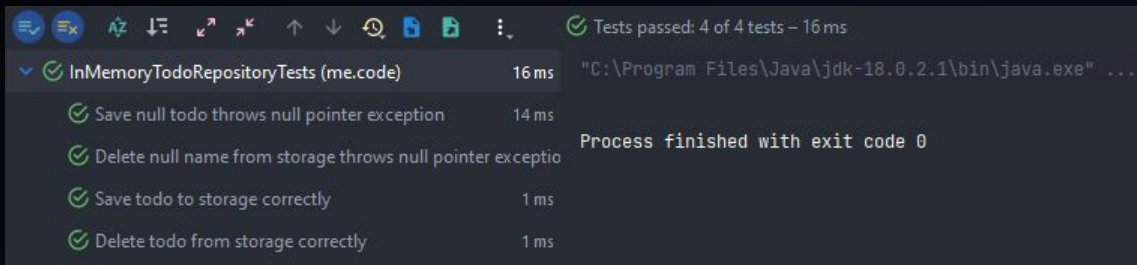
InMemoryTodoRepository

```
@Override
public void delete(String name) {
    if (name == null)
        throw new NullPointerException("'name' cannot be null");

    data.remove(name);
}
```

Steg 4 – Kör testet igen

InMemoryTodoRepositoryTests



The screenshot shows an IDE's test runner interface. At the top, a status bar indicates "Tests passed: 4 of 4 tests - 16 ms". Below this, a list of test results is displayed. The first entry is "InMemoryTodoRepositoryTests (me.code)" with a total duration of 16 ms. It is expanded to show four individual test cases, each marked with a green checkmark and its duration: "Save null todo throws null pointer exception" (14 ms), "Delete null name from storage throws null pointer exceptio" (1 ms), "Save todo to storage correctly" (1 ms), and "Delete todo from storage correctly" (1 ms). To the right of the test list, a message states "Process finished with exit code 0". The background of the IDE window is dark, and the test results are highlighted in a lighter shade.

Test Name	Duration
InMemoryTodoRepositoryTests (me.code)	16 ms
Save null todo throws null pointer exception	14 ms
Delete null name from storage throws null pointer exceptio	1 ms
Save todo to storage correctly	1 ms
Delete todo from storage correctly	1 ms

Process finished with exit code 0

Steg 5 & 6 – Förbättra och testa igen

- Finns inget uppenbart att förbättra

Steg 7 – Repetera för nästa test

- Spara todos ✓
 - Om ett null värde sparas skall 'NullPointerException' kastas ✓
- Radera todos ✓
 - Om namnet är null skall 'NullPointerException' kastas ✓
 - Om en todo inte finns skall 'TodoDoesNotExist' kastas

Steg 1 – Skriv test

InMemoryTodoRepositoryTests


```
@Test
@DisplayName("Delete todo that does not exist throws exception")
void delete_non_existing_todo_throws_exception() {
    // given
    var todo = new Todo( name: "Städa", description: "Städa mitt rum kl 15:00.");
    repository.save(todo);

    // when/then
    Assertions.assertThrows(
        TodoDoesNotExistException.class,
        () → repository.delete( name: "Handla")
    );

    // then
    Assertions.assertEquals( expected: 1, repository.count());
}
```

Steg 2 – Kör testet

InMemoryTodoRepositoryTests



```
Tests failed: 1, passed: 4 of 5 tests - 18 ms

InMemoryTodoRepositoryTests (me.code) 18 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
  ✓ Save null todo throws null pointer exception 14 ms
  ✗ Delete todo that does not exist throws exception 2 ms
    org.opentest4j.AssertionFailedError: Expected me.code.
    <4 internal lines>
    at me.code.InMemoryTodoRepositoryTests.delete_non_
    at java.base/java.util.ArrayList.forEach(ArrayList
    at java.base/java.util.ArrayList.forEach(ArrayList
  ✓ Delete null name from storage throws null pointer ex 1 ms
  ✓ Save todo to storage correctly 1 ms
  ✓ Delete todo from storage correctly
```

Steg 3 – Lös testet

InMemoryTodoRepository

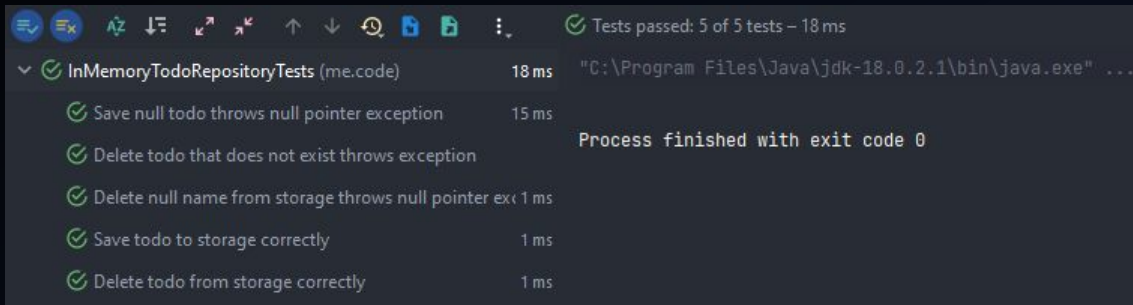
```
@Override
public void delete(String name) {
    if (name == null)
        throw new NullPointerException("'name' cannot be null");

    if (!data.containsKey(name))
        throw new TodoDoesNotExistException();

    data.remove(name);
}
```

Steg 4 – Kör testet igen

InMemoryTodoRepositoryTests



The screenshot shows an IDE's test runner interface. At the top, a status bar indicates "Tests passed: 5 of 5 tests - 18 ms". Below this, a list of test results is displayed. The first entry is "InMemoryTodoRepositoryTests (me.code)" with a duration of "18 ms" and a command line snippet. It is expanded to show five sub-tests, all of which passed. The sub-tests and their durations are: "Save null todo throws null pointer exception" (15 ms), "Delete todo that does not exist throws exception" (1 ms), "Delete null name from storage throws null pointer exception" (1 ms), "Save todo to storage correctly" (1 ms), and "Delete todo from storage correctly" (1 ms). A message "Process finished with exit code 0" is also visible.

Test Name	Duration
InMemoryTodoRepositoryTests (me.code)	18 ms
Save null todo throws null pointer exception	15 ms
Delete todo that does not exist throws exception	1 ms
Delete null name from storage throws null pointer exception	1 ms
Save todo to storage correctly	1 ms
Delete todo from storage correctly	1 ms

Process finished with exit code 0

Steg 5 – Förbättra koden

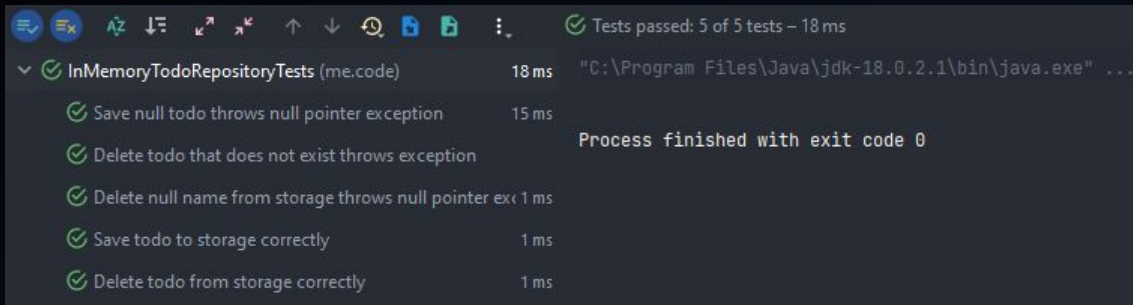
InMemoryTodoRepository

```
@Override
public void delete(String name) {
    if (name == null)
        throw new NullPointerException("'name' cannot be null");

    if (data.remove(name) == null)
        throw new TodoDoesNotExistException();
}
```

Steg 6 – Testa igen

InMemoryTodoRepositoryTests



The screenshot shows a test runner interface with a toolbar at the top containing icons for test execution, search, and other IDE functions. The status bar at the top right indicates "Tests passed: 5 of 5 tests - 18 ms". The main area displays a list of tests under the heading "InMemoryTodoRepositoryTests (me.code)". Each test is preceded by a green checkmark icon. The tests and their durations are: "Save null todo throws null pointer exception" (15 ms), "Delete todo that does not exist throws exception" (1 ms), "Delete null name from storage throws null pointer exception" (1 ms), "Save todo to storage correctly" (1 ms), and "Delete todo from storage correctly" (1 ms). The command line at the bottom shows the execution path: "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...

Test Name	Duration
Save null todo throws null pointer exception	15 ms
Delete todo that does not exist throws exception	1 ms
Delete null name from storage throws null pointer exception	1 ms
Save todo to storage correctly	1 ms
Delete todo from storage correctly	1 ms

Process finished with exit code 0

Steg 7 – Klar med repository, fortsätt till service

- Spara todos ✓
 - Om ett null värde sparas skall 'NullPointerException' kastas ✓
- Radera todos ✓
 - Om namnet är null skall 'NullPointerException' kastas ✓
 - Om en todo inte finns skall 'TodoDoesNotExist' kastas ✓

Steg 1 – Skriv test

TodoServiceTests

```
TodoRepository repository;
TodoService service;

@BeforeEach
void setup() {
    repository = Mockito.mock(TodoRepository.class);
    service = new TodoService(repository);
}

@Test
@DisplayName("Create todo stores in repository")
void create_stores_todo() {
    // given
    var name = "Städa";
    var description = "Städa mitt rum kl 15:00.";

    // when/then
    var todo :Todo = Assertions.assertDoesNotThrow(() -> service.create(name, description));

    // then
    Mockito.verify(repository).save(Mockito.any(Todo.class));
    Assertions.assertNotNull(todo);
}
```

Steg 2 – Kör testet

ToDoServiceTests

```
Tests failed: 1 of 1 test - 354 ms
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...

▼ ⚠️ ToDoServiceTests (me.code) 354 ms
  ⚠️ Create todo stores in repository 354 ms

    Wanted but not invoked:
    todoRepository.save(<any me.code.data.TODO>);
    -> at me.code.ToDoServiceTests.create_stores_todo(ToDoServiceTests.java:35)
    Actually, there were zero interactions with this mock.
```

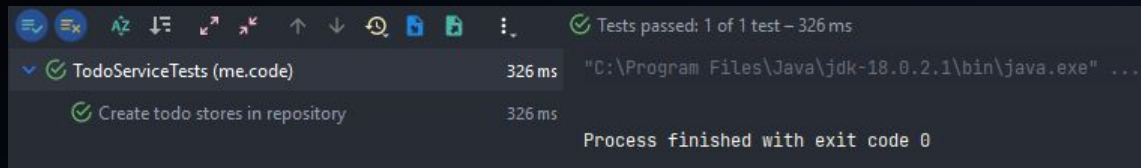
Steg 3 – Lös testet

TodoService

```
public class TodoService {  
  
    private final TodoRepository repository;  
  
    public TodoService(TodoRepository repository) { this.repository = repository; }  
  
    public Todo create(String name, String description) {  
        var todo = new Todo(name, description);  
        repository.save(todo);  
  
        return todo;  
    }  
  
    public Todo delete(String name) {  
        return null;  
    }  
  
}
```

Steg 4 – Kör testet igen

ToDoServiceTests



The screenshot shows a test runner interface with a toolbar at the top containing icons for test execution and navigation. The main area displays the test results for 'ToDoServiceTests (me.code)'. The test 'Create todo stores in repository' is shown as passed with a green checkmark and a duration of 326 ms. The command line path for the test is visible as '"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...'. At the bottom, a message states 'Process finished with exit code 0'.

```
Tests passed: 1 of 1 test - 326 ms
```

Test Name	Duration	Command
ToDoServiceTests (me.code)	326 ms	"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
Create todo stores in repository	326 ms	

Process finished with exit code 0

Steg 5 & 6 – Förbättra och testa igen

- Finns inget uppenbart att förbättra

Steg 7 – Repetera för nästa test

- Skapa todos ✓
 - Om en todo skapas dubbelt skall 'TodoAlreadyExistsException' kastas
 - Om en todo skapas utan namn skall 'TodoMustHaveNameException' kastas
- Radera todos
 - Om en todo inte finns skall 'TodoDoesNotExistException' kastas
- Kopplas till TodoRepository

Steg 1 – Skriv test

TodoServiceTests

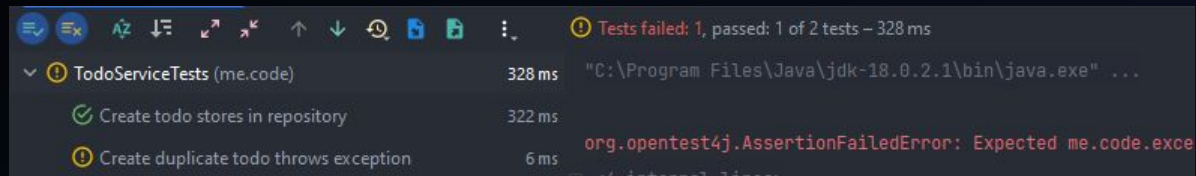
```
@Test
@DisplayName("Create duplicate todo throws exception")
void create_duplicate_throws_exception() {
    // given
    var name = "Städa";
    var description = "Städa mitt rum kl 15:00.";
    var todo = new Todo(name, description);
    Mockito.when(repository.contains(name)).thenReturn(true);
    Mockito.when(repository.getByName(name)).thenReturn(Optional.of(todo));

    // when/then
    Assertions.assertThrows(
        TodoAlreadyExistsException.class,
        () -> service.create(name, description: "any")
    );

    // then
    Mockito.verify(repository, Mockito.never()).save(Mockito.any(Todo.class));
}
```

Steg 2 – Kör testet

TodoServiceTests



```
Tests failed: 1, passed: 1 of 2 tests - 328 ms
▼ [info] TodoServiceTests (me.code) 328 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
  [pass] Create todo stores in repository 322 ms
  [fail] Create duplicate todo throws exception 6 ms org.opentest4j.AssertionFailedError: Expected me.code.exce
```

Steg 3 – Lös testet

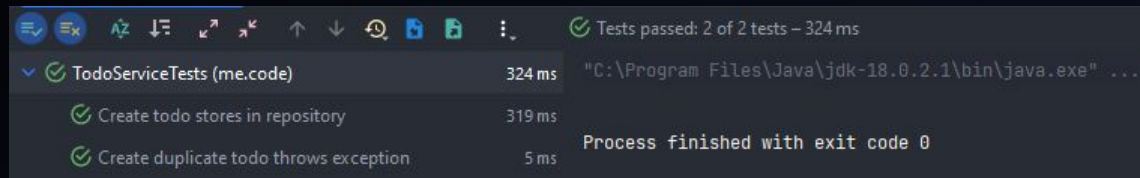
TodoService

```
public Todo create(String name, String description) {  
    if (repository.contains(name))  
        throw new TodoAlreadyExistsException();  
  
    var todo = new Todo(name, description);  
    repository.save(todo);  
  
    return todo;  
}
```

```
public interface TodoRepository {  
    void save(Todo todo);  
    void delete(String name);  
    boolean contains(String name);  
    Optional<Todo> getByName(String name);  
    int count();  
}
```

Steg 4 – Kör testet igen

ToDoServiceTests



Steg 5 & 6 – Förbättra och testa igen

- Finns inget uppenbart att förbättra

Steg 7 – Repetera för nästa test

- Skapa todos ✓
 - Om en todo skapas dubbelt skall 'TodoAlreadyExistsException' kastas ✓
 - Om en todo skapas utan namn skall 'TodoMustHaveNameException' kastas
- Radera todos
 - Om en todo inte finns skall 'TodoDoesNotExistException' kastas
- Kopplas till TodoRepository

Gör TDD processen för 'contains' och 'getByName'

```
@Override
public boolean contains(String name) {
    if (name == null)
        throw new NullPointerException("'name' cannot be null");

    return data.containsKey(name);
}
```

```
@Override
public Optional<Todo> getByName(String name) {
    if (name == null)
        throw new NullPointerException("'name' cannot be null");

    var todo :Todo = data.get(name);
    return Optional.ofNullable(todo);
}
```


Steg 1 – Skriv test

TodoServiceTests

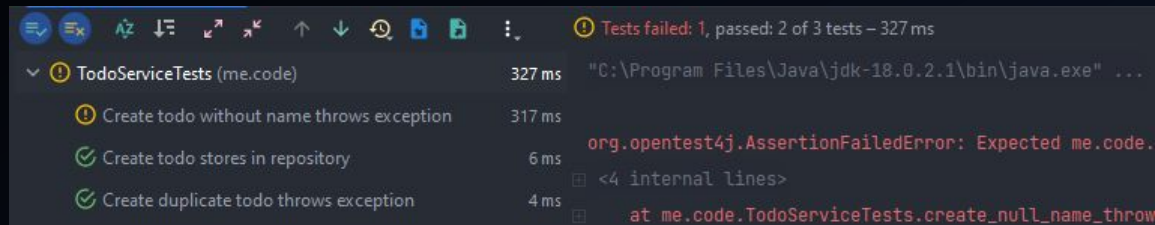
```
@Test
@DisplayName("Create todo without name throws exception")
void create_null_name_throws_exception() {
    // given
    var description = "Städa mitt rum kl 15:00.";

    // when/then
    Assertions.assertThrows(
        TodoMustHaveNameException.class,
        () → service.create( name: null, description)
    );

    // then
    Mockito.verify(repository, Mockito.never()).save(Mockito.any(Todo.class));
}
```

Steg 2 – Kör testet

ToDoServiceTests



```
Tests failed: 1, passed: 2 of 3 tests - 327 ms

▼ ⚠️ ToDoServiceTests (me.code) 327 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
  ⚠️ Create todo without name throws exception 317 ms
    org.opentest4j.AssertionFailedError: Expected me.code.
    <4 internal lines>
    at me.code.ToDoServiceTests.create_null_name_throw

  ✅ Create todo stores in repository 6 ms

  ✅ Create duplicate todo throws exception 4 ms
```

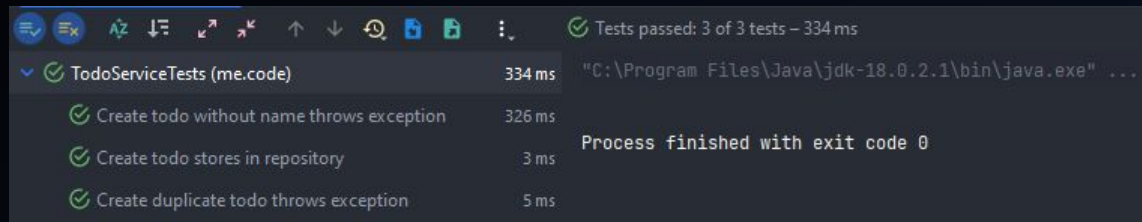
Steg 3 – Lös testet

TodoService

```
public Todo create(String name, String description) {  
    if (repository.contains(name))  
        throw new TodoAlreadyExistsException();  
  
    if (name == null)  
        throw new TodoMustHaveNameException();  
  
    var todo = new Todo(name, description);  
    repository.save(todo);  
  
    return todo;  
}
```

Steg 4 – Kör testet igen

ToDoServiceTests



Steg 5 & 6 – Förbättra och testa igen

- Finns inget uppenbart att förbättra

Steg 7 – Repetera för nästa test

- Skapa todos ✓
 - Om en todo skapas dubbelt skall 'TodoAlreadyExistsException' kastas ✓
 - Om en todo skapas utan namn skall 'TodoMustHaveNameException' kastas ✓
- Radera todos
 - Om en todo inte finns skall 'TodoDoesNotExistException' kastas
- Kopplas till TodoRepository

Steg 1 – Skriv test

TodoServiceTests

```
@Test
@DisplayName("Delete todo removes from storage")
void delete_todo_removes_storage() {
    // given
    var name = "Städa";
    var todo = new Todo(name, description: "Städa mitt rum kl 15:00.");
    Mockito.when(repository.contains(name)).thenReturn(true);
    Mockito.when(repository.getBy_name(name)).thenReturn(Optional.of(todo));

    // when/then
    var removed :Todo = Assertions.assertDoesNotThrow(() → service.delete(name));

    // then
    Mockito.verify(repository).delete(name);
    Assertions.assertNotNull(removed);
    Assertions.assertEquals(name, removed.getName());
    Assertions.assertEquals(todo.getDescription(), removed.getDescription());
    Assertions.assertEquals(todo.isCompleted(), removed.isCompleted());
}
```

Steg 2 – Kör testet

TodoServiceTests



```
Tests failed: 1, passed: 3 of 4 tests – 327 ms
▼ ⚠ TodoServiceTests (me.code) 327 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
  ✓ Create todo without name throws exception 314 ms
  ⚠ Delete todo removes from storage 9 ms
    Wanted but not invoked:
    todoRepository.delete("Städa");
    -> at me.code.TODOServiceTests.delete_todo_removes_sto
  ✓ Create todo stores in repository 2 ms
  ✓ Create duplicate todo throws exception 2 ms
```

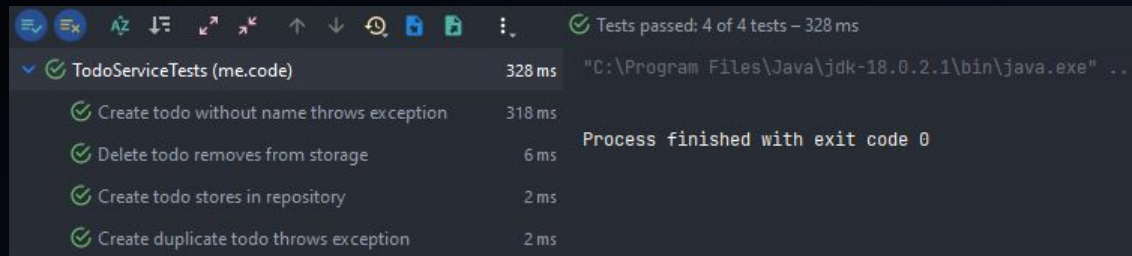

Steg 3 – Lös testet

TodoService

```
public Todo delete(String name) {  
    var todo : Optional<Todo> = repository.getByName(name);  
  
    repository.delete(name);  
  
    return todo.get();  
}
```

Steg 4 – Kör testet igen

ToDoServiceTests



The screenshot shows a test runner interface with a toolbar at the top containing icons for running, debugging, and other actions. The main area displays a table of test results for 'ToDoServiceTests (me.code)'. All four tests passed successfully, indicated by green checkmarks. The total execution time for the test suite is 328 ms. A message at the bottom right states 'Process finished with exit code 0'.

Tests passed: 4 of 4 tests – 328 ms	
▼ ✓ ToDoServiceTests (me.code)	328 ms
✓ Create todo without name throws exception	318 ms
✓ Delete todo removes from storage	6 ms
✓ Create todo stores in repository	2 ms
✓ Create duplicate todo throws exception	2 ms
Process finished with exit code 0	

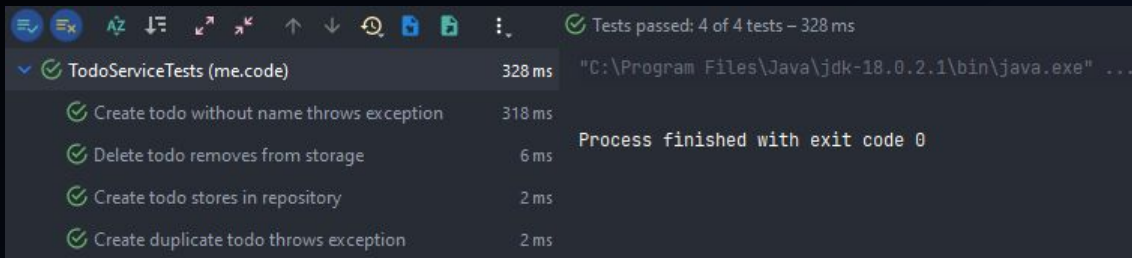
Steg 5 – Förbättra koden

TodoService

```
public Todo delete(String name) {  
    var todo : Optional<Todo> = repository.getByName(name);  
  
    repository.delete(name);  
  
    return todo.orElse( other: null);  
}
```

Steg 6 – Testa igen

ToDoServiceTests



The screenshot shows a test runner interface with a toolbar at the top containing icons for running, debugging, and other actions. The main area displays the test results for 'ToDoServiceTests (me.code)'. The tests are listed with green checkmarks indicating they passed. The total time for all tests is 328 ms. The command used to run the tests is shown as 'C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe' ... and the process finished with exit code 0.

Test Name	Duration
✓ Create todo without name throws exception	318 ms
✓ Delete todo removes from storage	6 ms
✓ Create todo stores in repository	2 ms
✓ Create duplicate todo throws exception	2 ms

Tests passed: 4 of 4 tests – 328 ms

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...

Process finished with exit code 0

Steg 7 – Repetera för nästa test

- Skapa todos ✓
 - Om en todo skapas dubbelt skall 'TodoAlreadyExistsException' kastas ✓
 - Om en todo skapas utan namn skall 'TodoMustHaveNameException' kastas ✓
- Radera todos ✓
 - Om en todo inte finns skall 'TodoDoesNotExistException' kastas
- Kopplas till TodoRepository

Steg 1 – Skriv test

TodoServiceTests

```
@Test
@DisplayName("Delete todo that does not exists throws exception")
void delete_non_existent_todo_throws_exception() {
    // given
    var name = "Städa";
    Mockito.when(repository.contains(name)).thenReturn(t: false);
    Mockito.when(repository.getByname(name)).thenReturn(t: Optional.empty());

    // when/then
    Assertions.assertThrows(
        TodoDoesNotExistException.class,
        () → service.delete(name)
    );

    // then
    Mockito.verify(repository, Mockito.never()).delete(name);
}
```

Steg 2 – Kör testet

TodoServiceTests



```
Tests failed: 1, passed: 4 of 5 tests - 346 ms

TodoServiceTests (me.code) 346 ms "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...

  Create todo without name throws exception 333 ms
  Delete todo removes from storage 8 ms
  Create todo stores in repository 2 ms
  Create duplicate todo throws exception 1 ms
  Delete todo that does not exists throws exception 2 ms

org.opentest4j.AssertionFailedError: Expected me.code.
<4 internal lines>
at me.code.TODOServiceTests.delete_non_existent_to
at java.base/java.util.ArrayList.forEach(ArrayList
at java.base/java.util.ArrayList.forEach(ArrayList
```

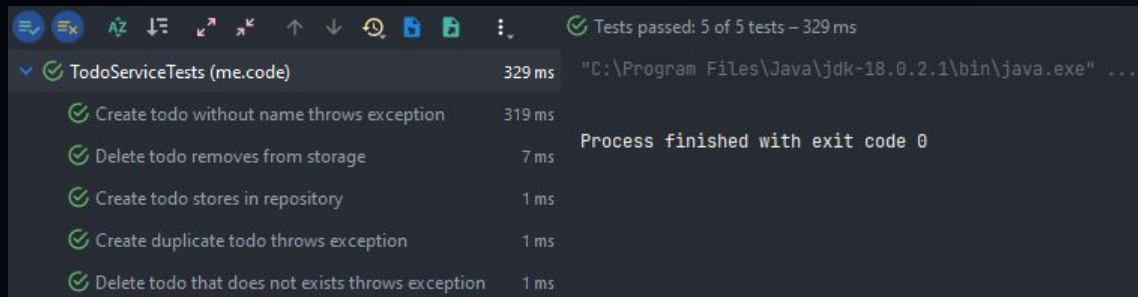
Steg 3 – Lös testet

TodoService

```
public Todo delete(String name) {  
    var todo :Todo = repository  
        .getByName(name)  
        .orElseThrow(TodoDoesNotExistException::new);  
  
    repository.delete(name);  
  
    return todo;  
}
```


Steg 4 – Kör testet igen

ToDoServiceTests



Steg 5 & 6 – Förbättra och testa igen

- Finns inget uppenbart att förbättra

Steg 7 – Repetera för nästa test

- Skapa todos ✓
 - Om en todo skapas dubbelt skall 'TodoAlreadyExistsException' kastas ✓
 - Om en todo skapas utan namn skall 'TodoMustHaveNameException' kastas ✓
- Radera todos ✓
 - Om en todo inte finns skall 'TodoDoesNotExistException' kastas ✓
- Kopplas till TodoRepository

Steg 1 – Skriv test

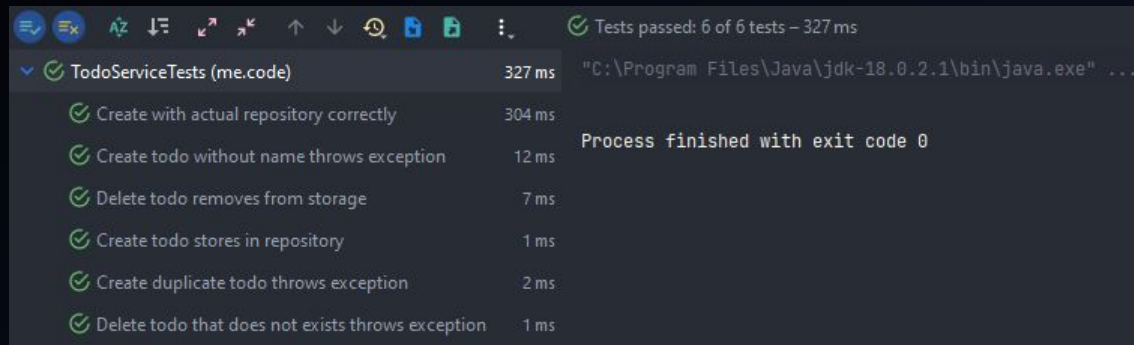
TodoServiceTests

```
@Test
@DisplayName("Create with actual repository correctly")
void integration_create() {
    // given
    var repository = new InMemoryTodoRepository();
    var service = new TodoService(repository);
    var name = "Städa";
    var description = "Städa mitt rum kl 15:00.";

    // when/then
    var todo :Todo = Assertions.assertDoesNotThrow(() → service.create(name, description));

    // then
    Assertions.assertNotNull(todo);
    Assertions.assertEquals( expected: 1, repository.count());
    Assertions.assertTrue(repository.contains(name));
}
```

Steg 2-, 3, 4, 5 & 6 – Kör testet. Det passerar.



The screenshot shows a test runner interface with a toolbar at the top containing icons for running, debugging, and other actions. Below the toolbar, a summary bar indicates "Tests passed: 6 of 6 tests - 327 ms". The main area displays a list of tests for "TodoServiceTests (me.code)". Each test is preceded by a green checkmark icon. The tests and their durations are: "Create with actual repository correctly" (304 ms), "Create todo without name throws exception" (12 ms), "Delete todo removes from storage" (7 ms), "Create todo stores in repository" (1 ms), "Create duplicate todo throws exception" (2 ms), and "Delete todo that does not exists throws exception" (1 ms). To the right of the test list, the text "Process finished with exit code 0" is displayed.

Test Name	Duration
✓ Create with actual repository correctly	304 ms
✓ Create todo without name throws exception	12 ms
✓ Delete todo removes from storage	7 ms
✓ Create todo stores in repository	1 ms
✓ Create duplicate todo throws exception	2 ms
✓ Delete todo that does not exists throws exception	1 ms

Process finished with exit code 0

Steg 7 – Repetera för nästa test

- Skapa todos ✓
 - Om en todo skapas dubbelt skall 'TodoAlreadyExistsException' kastas ✓
 - Om en todo skapas utan namn skall 'TodoMustHaveNameException' kastas ✓
- Radera todos ✓
 - Om en todo inte finns skall 'TodoDoesNotExistException' kastas ✓
- Kopplas till TodoRepository ✓

Fortsätt med samma sak på TodoController

- Förslagsvis unit tester först
- Följ upp med integration tester

Exempel på integration test med Spring Boot

```
@SpringBootTest
@TestPropertySource("classpath:application-test.properties")
@AutoConfigureMockMvc
class JavaSpringSpringTestingApplicationTests {

    @Autowired
    MockMvc mvc;

    @Test
    void testProductController() throws Exception {
        var builder : MockHttpServletRequestBuilder = put( urlTemplate: "/product/")
            .content("{\"name\": \"A name\", ...}")
            .contentType(MediaType.APPLICATION_JSON);

        mvc.perform(builder)
            .andExpect(status().isOk())
            .andExpect(content().json( jsonContent: "{\"}\"));
    }
}
```