

A mockolást legegyszerűbben úgy lehet megérteni, ha a BankAccount mellé teszünk egy külső függőséget, például egy értesítési szolgáltatást. A cél az lesz, hogy a BankService befizetés után küldjön értesítést, de a tesztben ne legyen valódi SMS vagy e-mail küldés. Ehelyett **Mockito** *mockot* használunk.

1. Először add hozzá a Mockito függőséget a pom.xml-hoz a JUnit mellé:

```
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-junit-jupiter</artifactId>
  <version>5.14.2</version>
  <scope>test</scope>
</dependency>
```

utána

```
mvn clean test
```

2. Hozd létre a NotificationService interfészt:

src/main/java/com/example/NotificationService.java

Tartalma:

```
package com.example;

public interface NotificationService {

    void send(String message);
}
```

Ez lesz a külső függőségünk.

3. Hozd létre a BankService osztályt:

src/main/java/com/example/BankService.java

Tartalma:

```
package com.example;

public class BankService {

    private final NotificationService notificationService;

    public BankService(NotificationService notificationService) {
        this.notificationService = notificationService;
    }
}
```

```
public void deposit(BankAccount account, double amount) {  
    account.deposit(amount);  
  
    notificationService.send("Deposit successful");  
}  
}
```

A fontos rész az, hogy a BankService nem hozza létre saját maga a NotificationService-t, hanem kívülről kapja meg. Ez a dependency injection egyszerű formája.

4. Most jön a mock teszt. Hozd létre:

src/test/java/com/example/BankServiceTest.java

Első változat:

```
package com.example;  
  
import org.junit.jupiter.api.Test;  
  
import static org.mockito.Mockito.mock;  
import static org.mockito.Mockito.verify;  
  
class BankServiceTest {  
  
    @Test  
    void depositSendsNotification() {  
  
        NotificationService notificationService =  
            mock(NotificationService.class);  
  
        BankService bankService =  
            new BankService(notificationService);  
  
        BankAccount account =  
            new BankAccount();  
  
        bankService.deposit(account, 1000);  
  
        verify(notificationService)  
            .send("Deposit successful");  
    }  
}
```

Ez már egy teljes Mockito teszt. A kulcssor:

```
mock(NotificationService.class)
```

Ez létrehoz egy `ál-NotificationService` objektumot. Nem küld semmit sehova. Csak “eljátssza”, hogy létezik. A másik fontos rész:

```
verify(notificationService)
    .send("Deposit successful");
```

Ez azt ellenőrzi, hogy a mock objektum `send()` metódusát valóban meghívták.

5. Futtasd:

```
mvn test
```

Ha jó, akkor a `BankServiceTest` is zöld lesz.

A kódot letöltheted

6. Most ellenőrizzük a számlaegyenleget is:

```
package com.example;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;

class BankServiceTest {

    @Test
    void depositSendsNotification() {

        NotificationService notificationService =
            mock(NotificationService.class);

        BankService bankService =
            new BankService(notificationService);

        BankAccount account =
            new BankAccount();

        bankService.deposit(account, 1000);

        assertEquals(
            1000.0,
            account.getBalance()
        );

        verify(notificationService)
            .send("Deposit successful");
    }
}
```

Itt már két dolgot tesztlünk: BankAccount állapota + NotificationService meghívása

7. Nézzük meg, mit jelent a mockolás gondolatilag.

Normál programban: BankService -> NotificationService -> valódi email / SMS / API

Tesztben: BankService -> MOCK NotificationService

Így a teszt gyors, izolált, és nem használ külső rendszert.

8. A következő fontos Mockito fogalom a when(...).thenReturn(...)

Ehhez csináljunk egy új interfészt: src/main/java/com/example/ExchangeRateService.java

```
package com.example;

public interface ExchangeRateService {

    double getRate(String from, String to);

}
```

Most a ExchangeRateServiceTest.java tesztfájlban:

```
ExchangeRateService rateService =
    mock(ExchangeRateService.class);

when(
    rateService.getRate("EUR", "HUF")
).thenReturn(400.0);
```

Ez azt jelenti: ha valaki ezt hívja:

```
getRate("EUR", "HUF")
```

akkor a mock válasza legyen:

400.0

Ehhez kell az import:

```
import static org.mockito.Mockito.when;
```

9. A három legfontosabb Mockito elem tehát:

```
mock(...)
```

ál objektum létrehozása.

```
when(...).thenReturn(...)
```

megmondjuk, mit válaszoljon.

```
verify(...)
```

ellenőrizzük, hogy meghívták-e.

10. Következő lépésként érdemes ezt a példát továbbvinni @Mock és @InjectMocks használatával, mert valódi projektekből gyakran így írják a Mockito teszteket:

```
@Mock
private NotificationService notificationService;

@InjectMocks
private BankService bankService;
```

Ez tisztább, mint minden tesztben kézzel meghívni a mock()-ot.

From:

<https://edu.iit.uni-miskolc.hu/> - Institute of Information Science - University of Miskolc

Permanent link:

<https://edu.iit.uni-miskolc.hu/tanszek:oktatas:mockito>

Last update: **2026/10/01 12:54**

