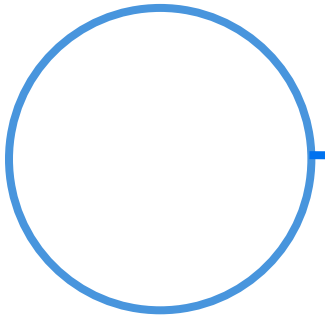
Szoftvertesztelés | 3-4. gyakorlat

Mockolás

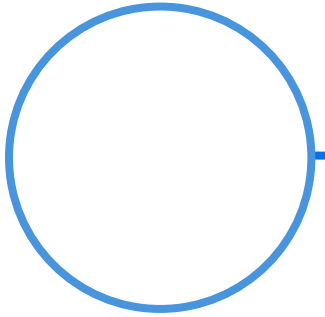
9 October 2025

Bemutatkozás
Előadók



Höltzl Tamás

Tesztmenedzser



Masa Tibor

Teszt architect

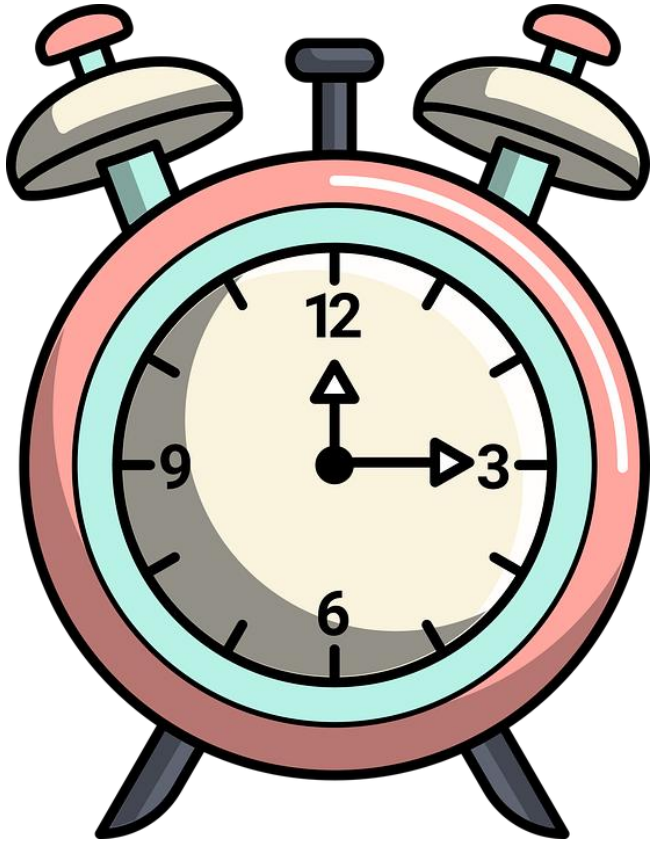
Bemutatkozás

ZDI – Carl Zeiss Digitial Innovation Hungary

Mielőtt elkezdjük…
Óra menete, kérdések

Menetrend

- 10:00 – 11:30
  - Mockolás alapjai
- 11:30 – 12:15
  - Ebédszünet
- 12:15 – 13:45
  - Mockolás külső függőségekkel



Agenda

01 Ismétlés

02 Környezet beállítása és Maven

03 Mockolás alapjai

04 Komplex mockolás

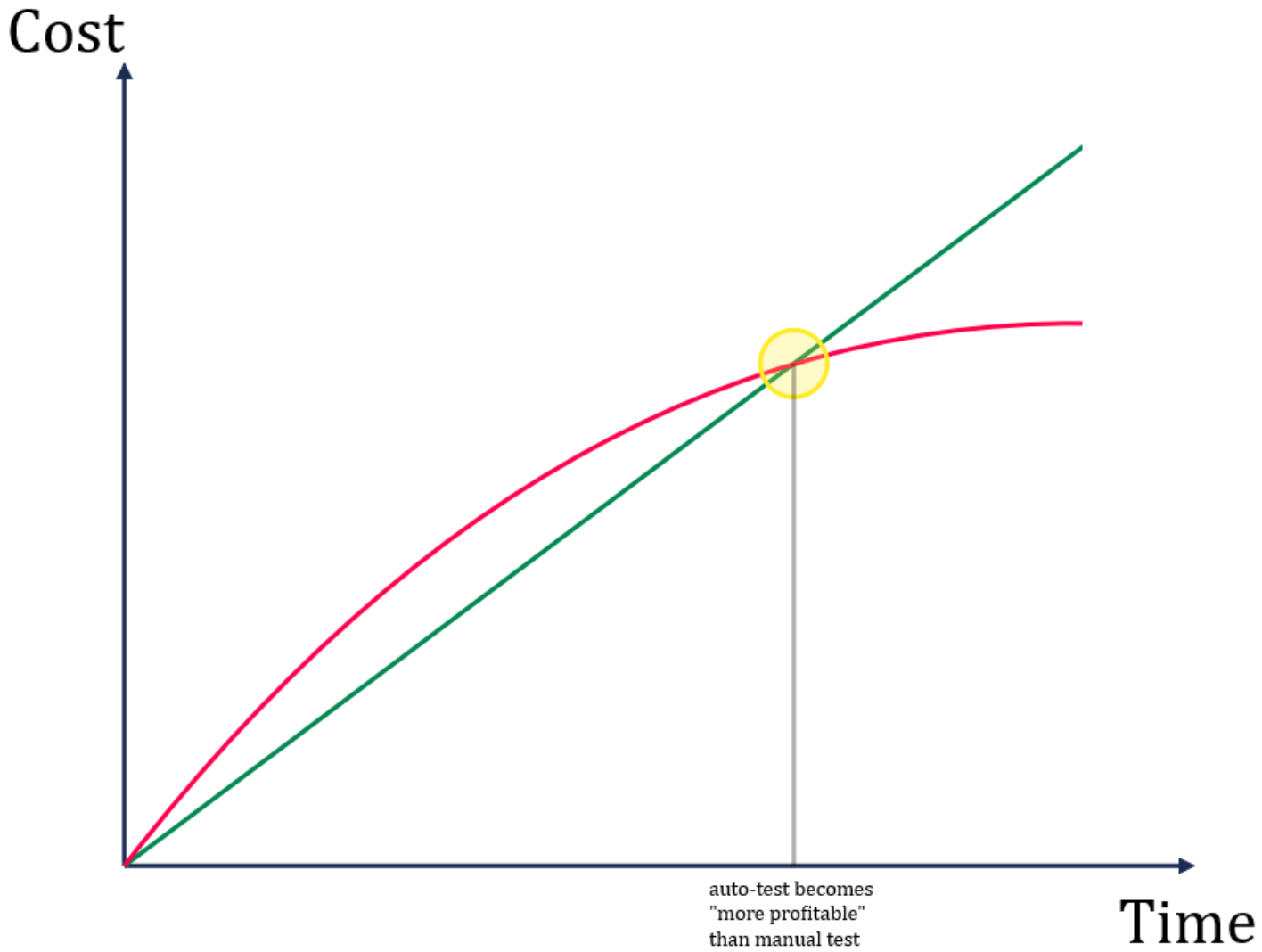
Definíció

Mi az a teszt automatizálás?

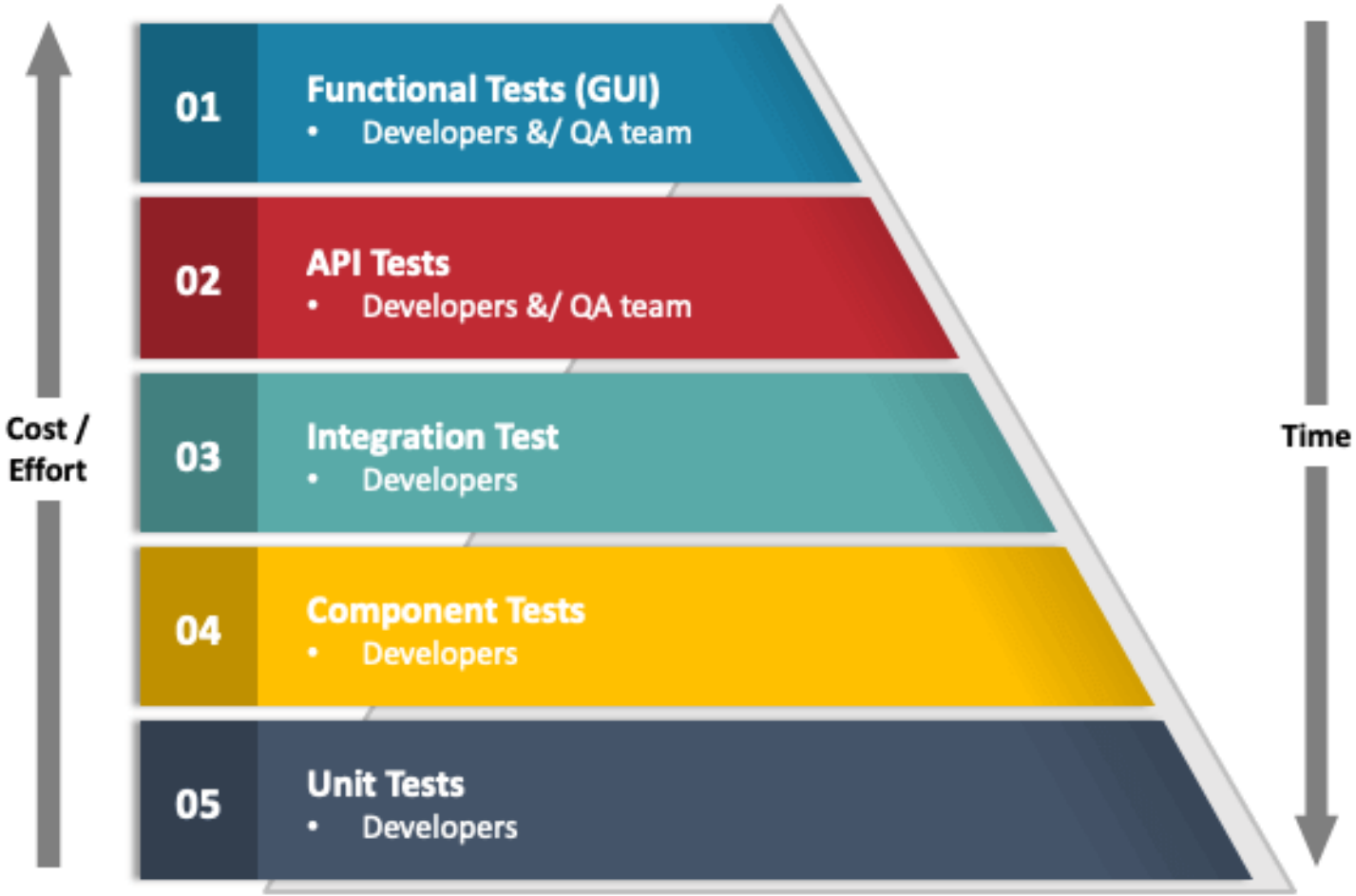
Speciális szoftverek használata

- tesztesetek futtatására
- az elvárt és aktuális eredmények összehasonlítására
- bármilyen egyéb, teszteléssel összefüggő tevékenység automatizálására

Manuális Vs automatizált tesztelés

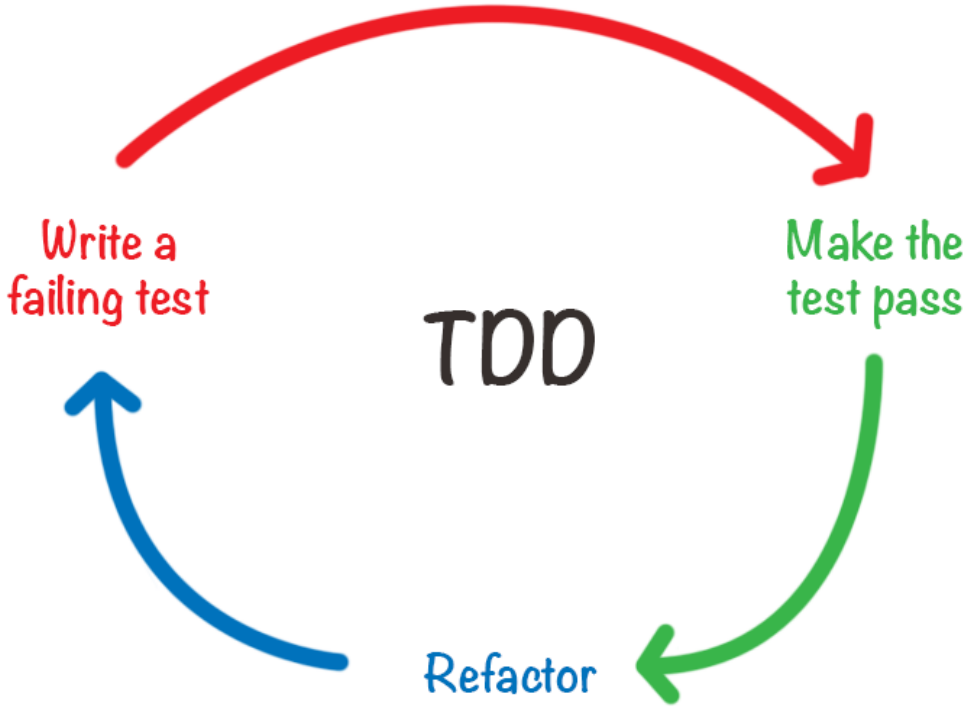


Általános bevezető
Teszt piramis



Forrás:: https://www.sketchbubble.com/en/presentation-testing-pyramid.html

Test-driven development



Agenda

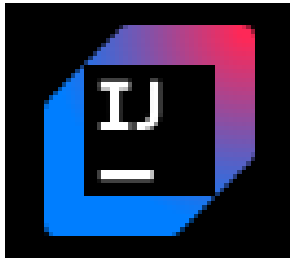
01 Elméleti alapok

02 Környezet beállítása és Maven

03 Mockolás alapjai

04 Komplex mockolás

Előkészületek 1.
Mire lesz szükségünk?



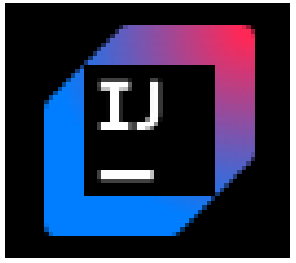
Előkészületek 2.
Környezet ellenőrzése

1. Windows command line: mvn -version

```
C:\Users>mvn -version
Apache Maven 3.8.4 (9b656c72d54e5bacbed989b64718c159fe39b537)
Maven home: C:\Program Files\Apache\apache-maven-3.8.4
Java version: 17.0.11, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17
Default locale: hu_HU, platform encoding: Cp1250
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users>
```

2. Alternatíva: installált IntelliJ community edition -> beépített Maven



Maven
A build folyamat szabványosítása

- Központi forrásból dolgozik -> **mvnrepository**
- „Convention over configuration”
- **POM** (project object model): A projekt, függőségeinek, valamint a build lépéseinek leírása
- A projekt gyökér könyvtárában
  - Kötelező elemek: modelVersion, groupId, artifactId, version
  - Opcionális elemek: packaging, dependencies, parent, modules, properties
- Függőségek letöltése a lokális **repository-ba**

```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/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>Software_Testing_25_26</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties...>

    <dependencies...>
</project>
```

Maven lifecycle

- **3 lifecycle:** default, clean, site -> benne fázisok, amelyeket **pluginek** futtatnak
  - Validate: projekt szerkezet ellenőrzés
  - Compile: Java kód -> bytekód
  - Test: Unit tesztek futtatása (Surefire)
  - Package: JAR / WAR, etc.
  - Verify: Minőségi ellenőrzés (Checkstyle, PMD)
  - Install: Telepítés a helyi repository-ba
  - Deploy: Feltöltés a távoli repository-ba
- **Parancsok** kiadásával futtatható a lifecycle
  - mvn clean install -> legtipikusabb parancs, output mappa törlése + teljes build folyamat install-ig
  - mvn package -> default a package fázisig

GIT
Saját repository használata

Projekt felmásolása saját repository-ba

- –git init
- –git add --all
- –git commit -m „Init project"
- –git branch -M main
- –git remote add origin https://github.com/username/repository.git
- –git push -u origin main

Lásd még: Adding locally hosted code to GitHub - GitHub Docs

Mockito telepítése

Maven repository

Nyisd meg a https://mvnrepository.com

Mockito Junit Jupiter

Keresd meg a legfrissebb mockito-junit-juper maven verziót

Pom.xml frissítése

A kapott kódot másold be a pom.xml **<dependencies>**-hez

```
<!-- https://mvnrepository.com/artifact/org.mockito/mockito-junit-jupiter -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>{mockito.version}</version>
    <scope>test</scope>
</dependency>
```

Agenda

01 Elméleti alapok

02 Környezet beállítása

03 Mockolás alapjai

04 Komplex mockolás

Mock, vagy soha

Definíció	■ A mock objektum egy tesztelt állapot ■ Valódi objektum helyettesítésére használva
Cél	■ Külső függőségek szimulálása ■ Tesztelt kód izolálása
Előnyök	■ Gyors tesztfuttatás ■ Könnyű hibakeresés ■ Determinisztikus teszt eredmények
Működése	■ Kód viselkedésének ellenőrzése ■ Valódi implementáció nélkül
Tipikus felhasználása	■ Adatbázis / külső API szimulálása ■ Idő- és hálózatfüggő logika kezelése
Eszköz	■ Mockito

Mockito alapok

Mock létrehozása	■ Mockito.mock metódussal vagy @Mock annotációval ■ *User user = Mockito.mock(User.class);* ■ *@Mock User mockUser;*
Viselkedés beállítása	■ when() és thenReturn() metódusokkal ■ *when(user.getId()).thenReturn(1);*
Hívás ellenőrzése	■ verify() metódussal ■ *verify(user).getUsername();*
Paraméterek helyettesítés	■ any(), anyInt(), stb. metódussal ■ *when(svc.getData(anyInt()).thenReturn(„mockData”);*
Metódus hívási számának ellenőrzése	■ times() metódussal ■ *verify(user, times(2)).getUsername();*
Kivételdobás ellenőrzése	■ doThrow() metódussal ■ *doThrow(new Exception()).when(user).getUsername();*

Mockito annotációk használata

@ExtendWith(MockitoExtension.class)

- Mockito használata a tesztfuttatáshoz
- Osztály vagy metódus szinten

@Mock

- Újrahasznosítható mock objektum létrehozása
- Konstruktor paramétert nem fogad!

```
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class TriangleMockTest {
    @Mock Triangle mockTriangle;
    @Test
    public void triangleMockTest() {
        //Triangle mockTriangle = Mockito.mock(Triangle.class)
        //Triangle mockTriangle = Mockito.mock(Triangle.class,
            withSettings().useConstructor(3,4,5));
    }
}
```

Kezdeti fájlok

https://tinyurl.com/miskolc2526

1. feladat – Triangle osztály tesztelése mock objektumokkal

Készítsünk egy mockolt objektumot a korábban létrehozott Triangle osztályból, majd ellenőrizzük azt

- *Létrehozás:* Mockito.mock
  - Három tetszőleges oldalhosszal
- *Viselkedések mockolása:* when() és thenReturn()
  - isIsosceles() és getPerimeter() ellenőrzése
- *Függvényhívások ellenőrzése:* verify(), times()
  - Ellenőrizzük, hogy a függvények meg lettek hívva
  - Ellenőrizzük, hogy a függvények a megfelelő számban lettek meghívva
- *Kivételkezelés ellenőrzése:* doThrow()
  - Egy nem implementált függvény ellenőrzése

Mockito - Függőségek

```
import org.mockito.junit.jupiter.MockitoExtension;

                    import static org.mockito.Mockito.*;
import static org.mockito.Mockito.doThrow;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
```

Agenda

01 Elméleti alapok

02 Környezet beállítása

03 Mockolás alapjai

04 Komplex mockolás

Mockolás hasznossága

- POJO esetén erősen korlátozott
  - Állapotot tárol, nincs külső függősége
- Hasznos, ha van külső függőség
  - Adatbázis, külső szolgáltatás, API, stb.
  - Valamit például egy adatbázisban akarok tárolni
  - Az én implementációm már kész, de az adatbázis még nem elérhető
- Hogyan tesztelem a működést, ha
  - az adatbázis nem elérhető?
  - nem elérhető adatot akarok lekérni?
  - túl lassú az adatbázis-lekérdezés?
- Megoldás: Mockoljuk az adatbázist!

Függőségek mockolása

@InjectMocks annotáció

- Egy osztály példányát automatikusan létrehozza, és injektálja a mockolt függőségeket.
  - megjegyzés: a @Mock annotációval ellátott függőségek lesznek injektálva
- Előnyei:
  - a tesztelt osztály valódi implementáció, de függőségei mockok lesznek!
  - jobb tesztelhetőség, nincs szükség manuális inicializációra
  - komplex hierarchia tesztelése karbantartható, bővíthető módon

Valós példa – klasszikus háromrétegű architektúra

- Domain / modell objektum: csak adatot és minimális belső logikát tartalmaz (pld. Table)
- Service / üzleti logika : szétválasztja a domain-t és a külső függőségeket, függősége az adatbázis (pld. TableService)
- Repository / adatelérési réteg: adatbázis / API / fájlrendszer réteg, ahonnan adatokat nyerünk ki. A domain objektumnak nem kell tudnia róla! (pld. TableRepository)

2. feladat – Table osztály tesztelése DB-réteg mockolásával

Bővítsük a korábban létrehozott Table osztályunkat egy Service és egy Repository osztállyal

- *TableService:*
  - egyetlen függőség, TableRepository
  - két metódus:
    - public int getTableCapacity(int id)
      - visszaadja, hányan ülhetnek le egy adott ID-jú asztalhoz, vagy hibát dob, ha adott ID-val nem található asztal az adatbázisban
      - A TableRepository findById(int id) függvényét hívja
  - public Table getLargestTable()
    - visszaadja a legnagyobb területű asztalt az adatbázisból, vagy hibát dob, ha nincs asztal az adatbázisban
    - a TableRepository findAll függvényét hívja
- *TableRepository:*
  - Az adatbázis kapcsolatot szimulálja, két függvénnyel, mindkettő UnsupportedOperationException-t dob híváskor
    - public List<Table> findAll()
    - public Rable findById(int id)

2. Feladat - implementáció

```java
public class TableService {
    3 usages
    private final TableRepository repository;

    no usages
    public TableService(TableRepository repository) {
        this.repository = repository;
    }

    no usages
    public int getTableCapacity(int id) {
        Table table = repository.findById(id);
        if (table == null) {
            throw new IllegalArgumentException("Table not found with id: " + id);
        }
        return table.getCapacity();
    }

    no usages
    public Table getLargestTable() {
        List<Table> tables = repository.findAll();
        if (tables == null || tables.isEmpty()) {
            throw new IllegalArgumentException("No tables found in repository");
        }
        Table largest = tables.getFirst();
        for (Table t : tables) {
            if (t.area() > largest.area()) {
                largest = t;
            }
        }
        return largest;
    }

}
```

```java
public class TableRepository {

    1 usage
    public Table findById(int id) {
        // Here we should have a DB call!
        throw new UnsupportedOperationException("not implemented");
    }

    1 usage
    public List<Table> findAll() {
        // Here we should have a DB call!
        throw new UnsupportedOperationException("not implemented");
    }
}
```

2. feladat

- **1) Teszteljük a TableService osztályt a TableRepository mockolásával**
  - Definiáljuk a TableRepository-t és a TableService-t a megfelelő annotációk használatával
  - Teszteljük mindkét TableService függvényt egy-egy pozitív és negatív ággal

- **Segítség:**
  - Tanult annotációk: @ExtendWith, @Mock, @InjectMocks
  - when() és thenReturn() függvénnyel definiálható, mit kellene csinálnia a mockolt osztálynak
  - assertThrows()-al kivétel dobását lehet ellenőrizni

Gyakorló feladat – User osztály tesztelése mockokkal

Készítsünk egy mockolt objekumot a korábban létrehozott User osztályból, majd ellenőrizzük azt

- A user az alábbi adatokat adja vissza:
  - userName = TesztElek
  - password = TOPSECRET
  - isLoggedIn = true
  - id = 123
- A verify() függvény használatával ellenőrizzük az egyes metódusok meghívását
- Írjunk egy UserDatabase osztályt, amely egy adatbázist szimulál
  - Egy függvénye legyen: setPassword(int id, String newPwd), és dobjon egy UnsupportedOperationException-t
  - A user osztály eddig nem implementált updatePwd függvénye hívja a UserDatabase függvényét
  - Segítség: ezúttal nincs köztes service osztály, a User-nek ismernie kell a repository-t!

Emlékeztető | Olvasnivaló

Mockito

- https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html
- https://site.mockito.org/
- https://www.baeldung.co
- https://www.tutorialspoint.com/mockito/index.htmm/mockito-series

Junit 5

- https://junit.org/junit5/docs/current/user-guide/
- https://www.baeldung.com/junit-5
- https://devqa.io/junit-5-annotations/

Maven

- https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
- https://www.tutorialspoint.com/maven/index.htm
- https://www.baeldung.com/maven