**2. modul**

Programming Basics

OOP Basics

oop basics, programming basics, java basics, java core, junit, sql basics

# Bevezetés

Object oriented programming, azaz Objektum Orientált Programozás egy olyan paradigma (alapelv rendszer), amely meghatározza milyen alapvető elvek szerint közelítik meg az adott feladat megoldását és a program felépítését, illetve hogyan épülnek fel és kapcsolódnak egymáshoz a kód elemei. (*Vannak más paradigmák is, pl funkcionális, procedurális vagy logikai programozás is)*

Előnyei

* *Komplex dolgokat lehet egyszerű, kisebb, könnyen reprodukálható struktúrákban modellezni*
* *A létrejött kód egyszerűen újrafelhasználható*
* *Gyorsabban futtatható programokat hozhatunk létre*
* *A kód struktúrája átlátható*
* *A fejlesztés üteme is gyorsabb, mert lehetővé teszi az osztályok párhuzamos fejlesztését*
* *A program egyszerűbben módosítható, debugolható, karbantartható*
* *Nagyobb biztonság* (az egységbezárásnak és az absztrakciónak köszönhetően védettebbek az adatok)

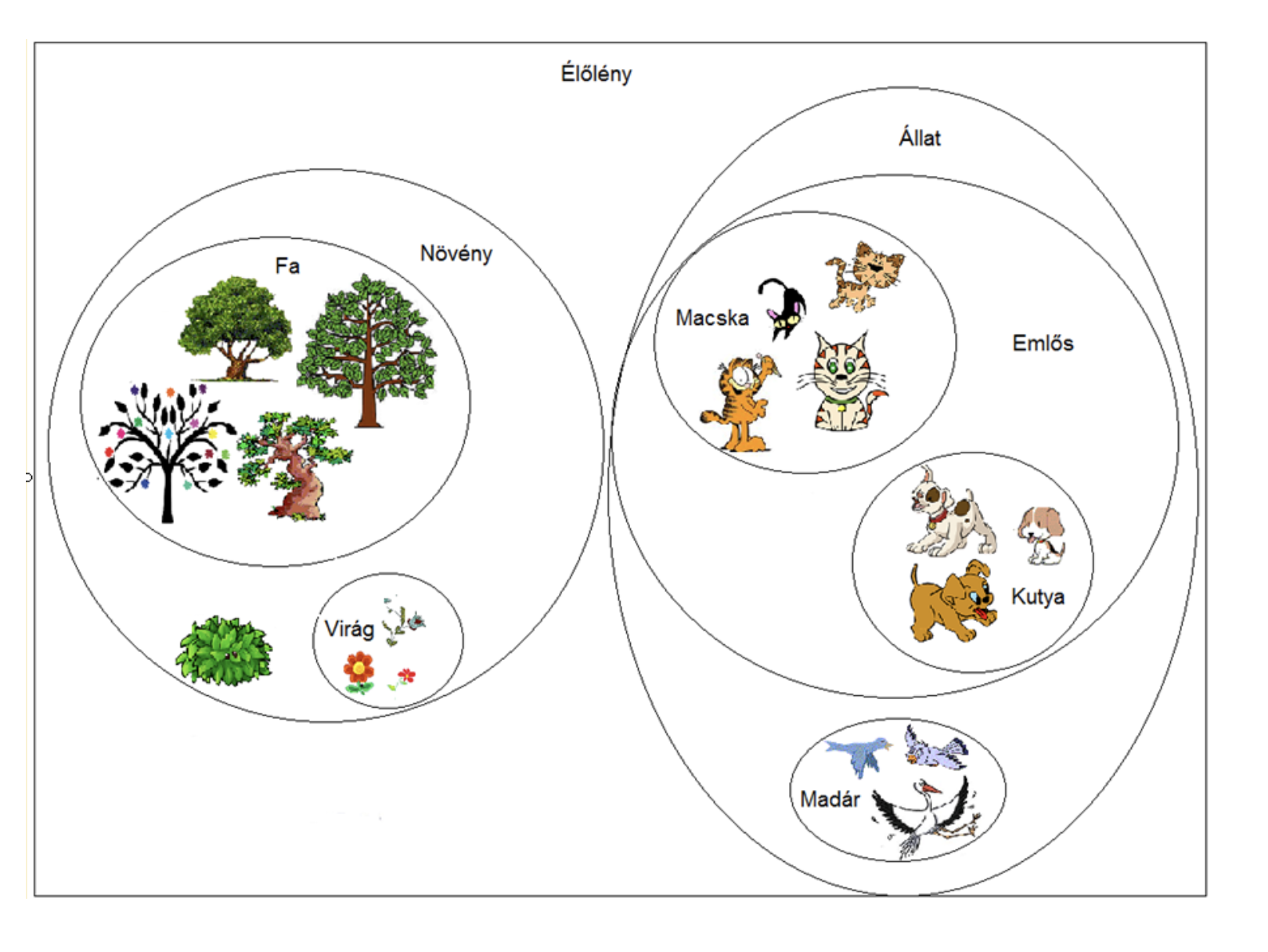
Legnépszerűbb magas szintű programozási nyelvek

* C++
* C[#](https://codeberryschool.com/blog/hu/a-c-sharp-programozas-alapjai/)
* **Java**
* Javascript
* Typescript
* Python

# Objektumok és osztályok

**Objektum (Object**), ahogy a való életben is maguk a kézzelfogható “tárgyak”, amelyeknek vannak jellemzői *(attribútumai)* és funkcióik *(metódusai)*

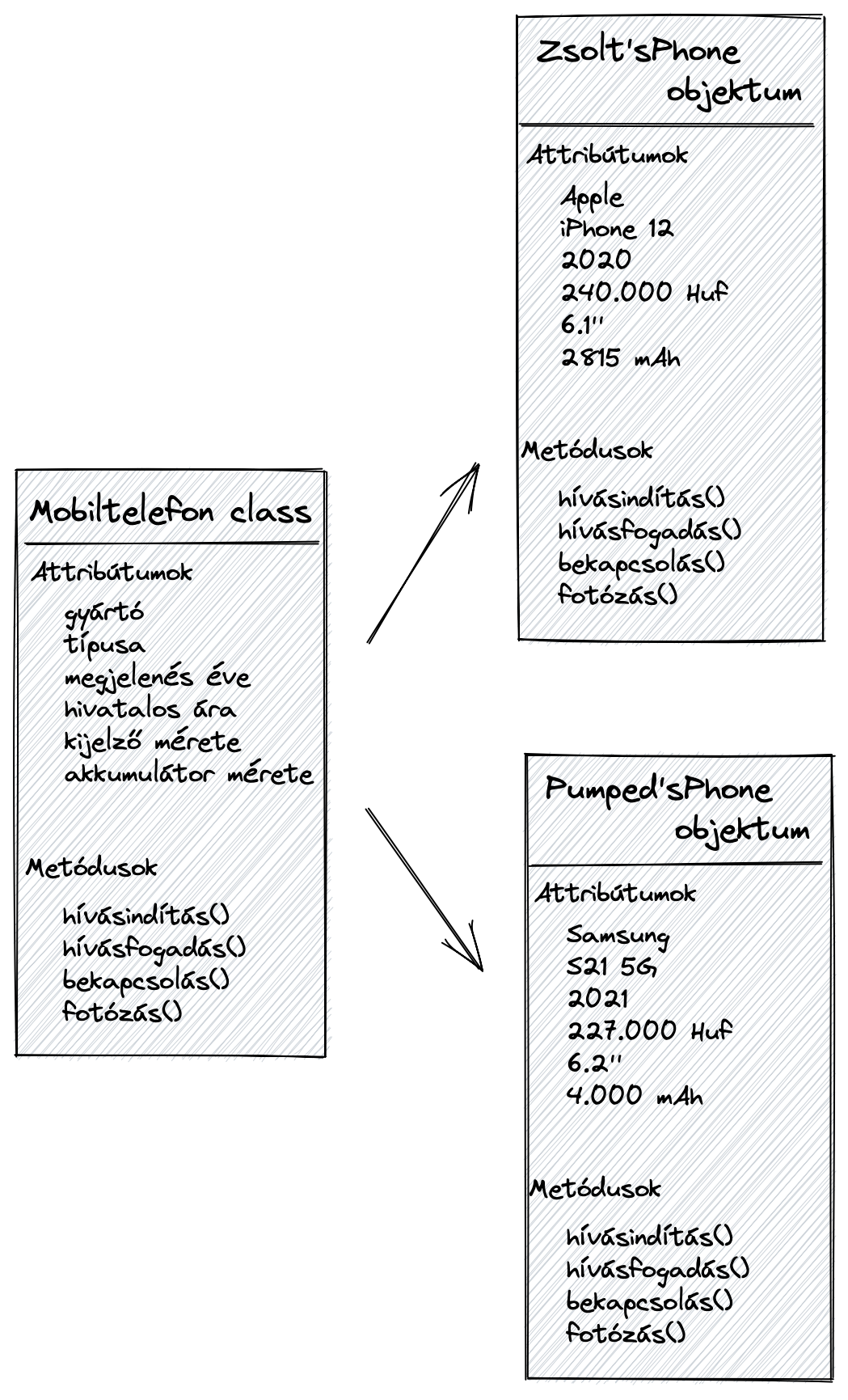
**Osztály (Class)**, maga a tervrajza, alaprajza az objektumoknak, ami azért jó, mert a használatával nem egyesével kell megmondani minden egyes létrehozott objektumról annak szerkezetét. Ezt bármikor újra és újra felhasználhatjuk és ez alapján akármennyi objektumot tudunk létrehozni.



## 

## 

## Például



## Példa kódból

<https://github.com/progmatictesztautomation23h2/oktatoi/blob/main/java/Learner.java>

# OOP alapelvek

## Egységbezárás (encapsulation)

Az objektum minden attribútuma és metódusa, az adott objektumon belül egy önálló egészt alkotva tárolódnak.

Más objektumok metódusai csak akkor tudják befolyásolni őket, ha ezt exakt módon megengedjük nekik.

**Például**

* Adott egy piros Porsche 911 autó objektum, aminek van gyártója, típusa és színe attribútumai vannak az osztályban definiálva. Ezeket az attribútumokat beállítjuk az osztályban, hogy ne legyenek módosíthatók. *(egy-egy plusz kulcsszóval)*. Így miután létrehoztuk nem lehet módosítani azt. Viszont ha van egy engedélyezett *motorindítás()* metódusa, akkor azt többször is meghívhatják.

## Absztrakció (abstraction)

Csak a szükséges információkat szolgáltatjuk a külvilágnak, elrejtjük a végrehajtás részleteit. Az absztrakció révén az objektumok csak a felszíni működésben, interakcióban fontos elemeket láttatják a többi objektummal, a belső struktúrát, ami mindehhez szükséges, elrejtik.

**Például**

* A fenti autós példában az engedélyezett *motorindítás()* metódus működése, hogy mi történik benne, az az objektumot használót nem érdekli, neki elég ha a Porsche 911 autó elindul a metódus hatására.

## Öröklés (inheritance)

Fejlesztés során minden lehetséges esetben meg kell próbáljuk elkerülni a kód duplikációt. Cél az újrafelhasználhatóság. Abban az esetben, amikor több osztály nagyon hasonló és vannak közös attribútumaik és/vagy metódusaik akkor alkalmazhatjuk az öröklést.

Fejlesztésben a szülő (parent) és gyerek (child) kifejezéseket használjuk ennek az interpretációjára, méghozzá úgy hogy a gyerek örököl a szülőtől.

Öröklés során a gyerek örökli a szülő logikáját és felépítését, hozzáférnek a szülő metódusaihoz.

**Például**

* Ha van a fenti példákban használt autó objektum, aminek már van gyártója, típusa, színe, akkor az új, gyerek objektumok megöröklik ezeket és létre tudunk hozni, egy “teherautó” vagy “busz” objektumot, amiknek szintén lesz gyártója, típusa, színe automatikusan.

## Polimorfizmus (polymorphism)

Megengedi, hogy a gyerek örökölt attribútumai és metódusai mellett saját metódusai legyenek oly módon, hogy képes legyen másképp meghatározni az abban lévő kódrészletet, más lehet az ő, saját implementációja.**Például**

* A *motorinditas()* metódus a busz esetén másképp működik, más utasításokat hajt végre, mint egy autó objektum esetén.

# UML

Unified Modeling Language (Egységesített Modellező Nyelv) Bővebb leírás: [link](https://hu.wikipedia.org/wiki/Unified_Modeling_Language)

Előnyei:

* Nyílt szabvány
* A szoftverfejlesztés ciklusainak mindegyikét támogatja
* Hatalmas tapasztalati tudásra épít
* Sok eszköz és sok cég támogatja, fejleszti a szabványt

Mire használjuk? Használhatjuk szoftverrendszer elemeinek:

* Vizualizálására
* Specifikálására
* Dokumentálására

Statikus modellek:

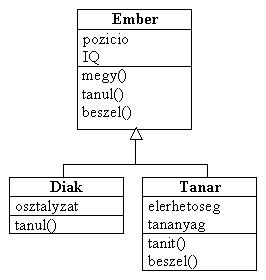
* Osztálydiagram
* Csomagdiagram
* Komponensdiagram

Dinamikus modellek:

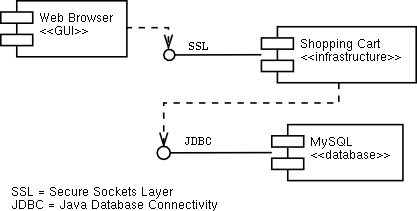
* Állapotdiagram
* Szekvenciadiagram
* Használati eset (use case) diagram

## 

## Osztálydiagram



## Komponens diagram



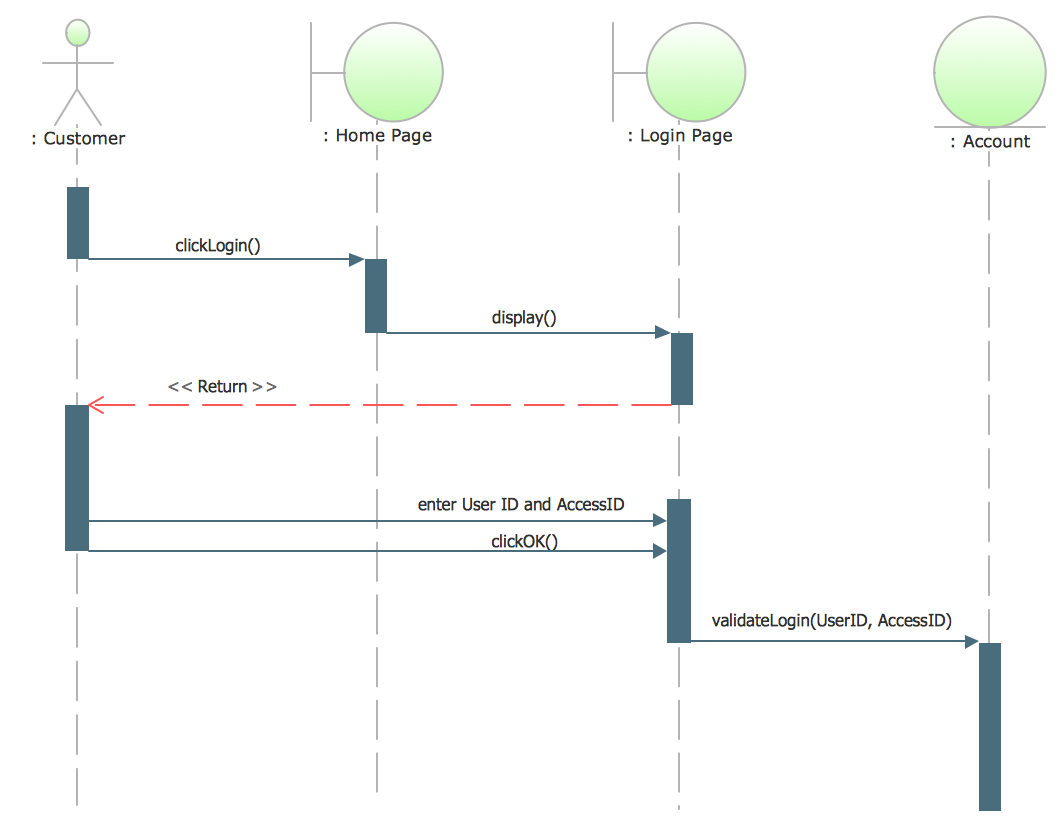
## 

## 

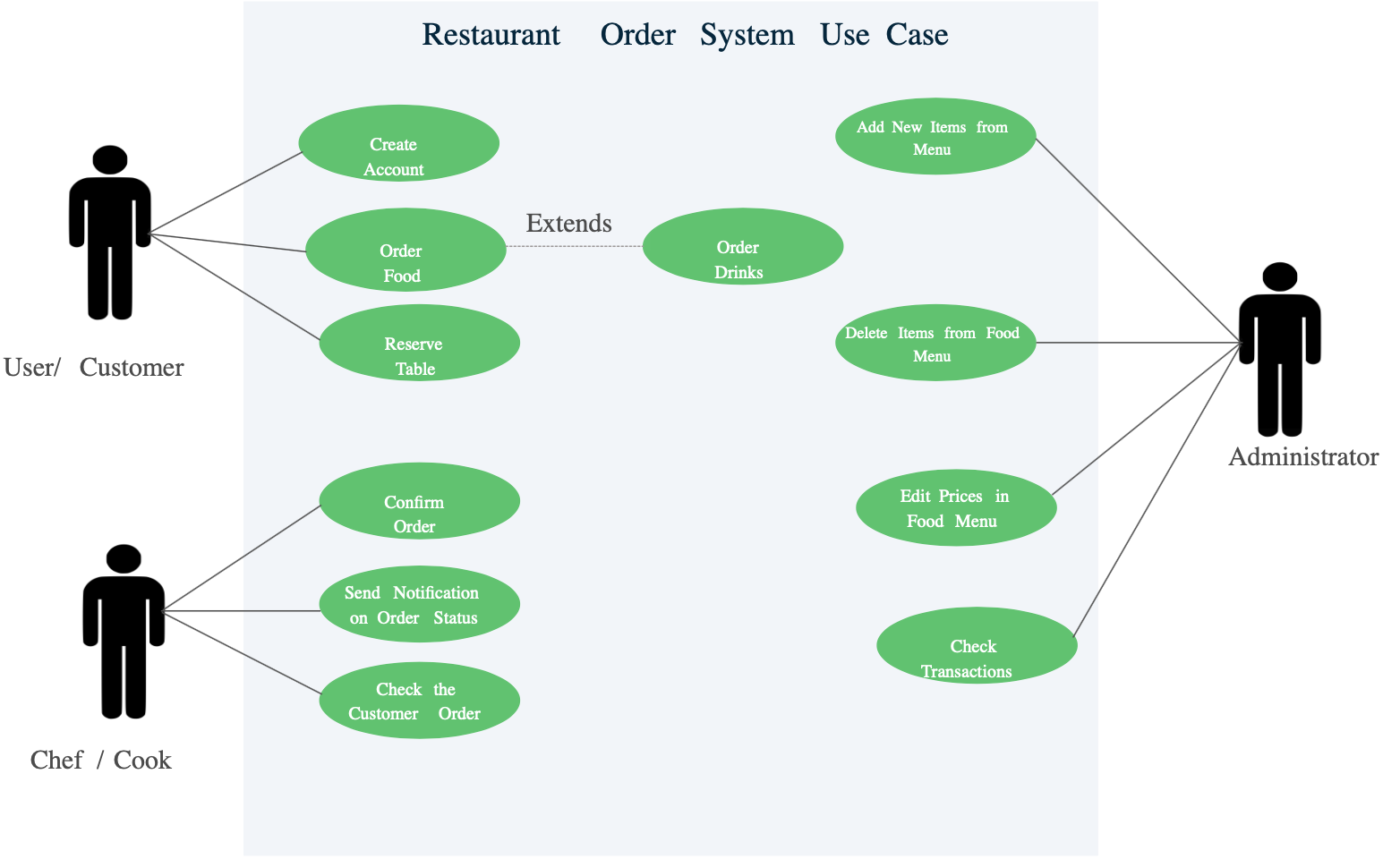
## Állapotdiagram

# 

## Szekvencia diagram



## Használati eset diagram



# Feladatok

1. (Órai) Nézzünk meg legalább 3 példát egy objektumra és hozzá tartozó osztályokra.
2. Készítsd el a Víz osztályt és hozzá legalább 2 hozzá tartozó objektumot önállóan <https://excalidraw.com/>-ban
3. (Órai) Készítsük el az autó-sportkocsi osztály diagramját, melyen látszik az öröklődés.
4. (Órai) Készítsük el a klasszikus fánk-csokis fánk, lekváros fánk osztály diagramját.
5. Készítsd el a Munkavállaló - Manager / Beosztott öröklés osztály diagramját.

# Linkek

Oracle doc: <https://docs.oracle.com/javase/tutorial/java/concepts/index.html>

JAVA OOP Doc: <https://www.oracle.com/java/technologies/oop.html>

Codeberry-s forrás anyag: <https://codeberryschool.com/blog/hu/mi-az-objektum-orientalt-programozas/>