Nesneye Yönelik Programlama

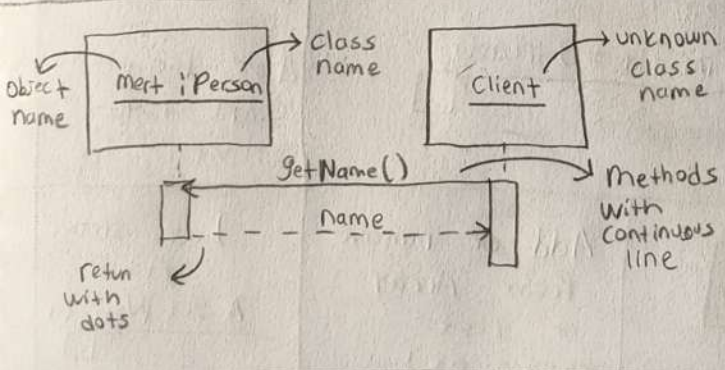
```
import java.util.Scanner
Scanner name = new Scanner(System.in)
Public Static Void main(String[] args){
int x = Integer.valueOf(variable) ⟶ Integer object → maybe better.
int y = Integer.ParseInt(variable) ⟶ Primitive, int → only can enter string
double result = (double) first / second
boolean a = first > second; → a = true.
            3     2
```

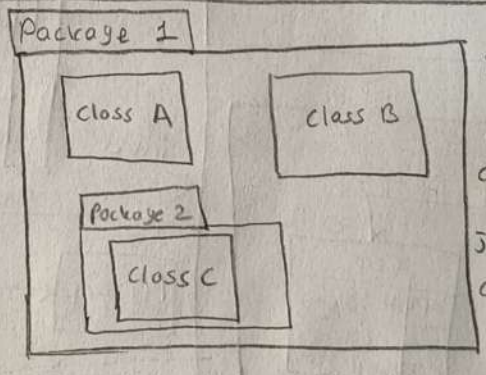
* Classlar atama yapilmazsa "null" olur. Primiteveler olmaz.

```
text.Equals("something")
String
ArrayList<String> list2 = new ArrayList<>();
          Object
for(String text: list2){ → Loop
}
list2.add("A"), list2.remove(0) or list2.remove("A"), list2.contains("A")
sout(list) = [A,B] > not same with Arrays.
             Output
int[] list = new int[x], x = size of Array
list.length();
String[] parts = text.split(",");
                 Object  Seperator > Splitters
char character = text.charAt(index)
@Ovverride
equals(); > Eger bu methodu ovverride yaparsak ArrayList'in contains methodu da degisir. Ayni methodu kullaniyor.
```

/* Commenting */

UML Diagrams






```
import Package1.C
Java.io.File > Sar Wo
com.Java.File
↓
Java\io\File.Javc
com\Java\File.Ja
```

```
import Package1.*; → import all classes from Package1 → A, B
       all → doesnt include Package2 *
import Package1.Package2.Class C; or .*;
```

ClassName
- PrivateField : Type
+ PublicVoidMethod()
+ PublicMethod() : ReturnType
+ ParamMethod(Param : ParamType)
+ main(String[])

Car
- plate : String
+ getPlate() : String
+ setPlate(String)

Static members → Accessed from class name: Not object ★

ClassName.memberName

Static Methods → ClassName.method()

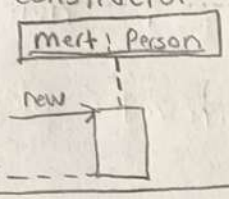
UML → aMember : Type {final, static}

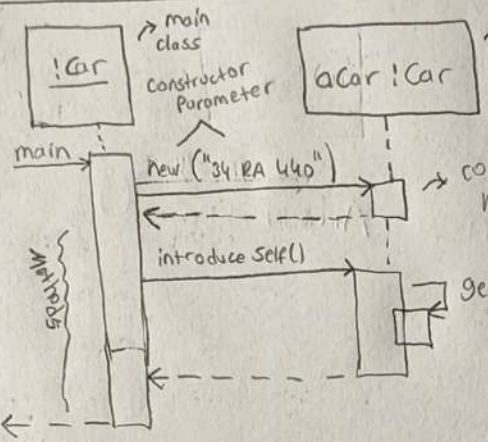
Static = Shared usage

Constructors with different amount/type of Parameters → Overloading ★

Wrapper → turns primitive to a Object.

Constructor





ArrayList<Class>()

↳ Generic

Yalnızca o class'ın veri tipini tutabilir. örn String.

ArrayList<>()

↳ Hersey tutar

~ Not ^ XOR >> << bit shifter

Math.Random() : 0-1 Random number

Math → abs(1.1), max, min

Ceil(3.1 → 4), floor(3.9 → 3)

round(3.5 → 4, 3.1 → 3)

sqrt → √

Array size = a.length

ArrayList size = a.size()

Not length() X

scanner.next() → reads first word

scanner.nextInt → reads first int than leave "\n" in buffer. have to use empty scanner after that.

Java.util.Random

↳ Random random = new Random()

random.nextInt()

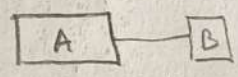
Primitives...

random.nextInt(5)

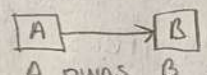
↳ [0,5)

ArrayList sadece Class tutar. Primitive tutmaz!

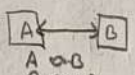
Relations between Objects



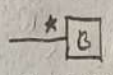
Association



A owns B



A and B

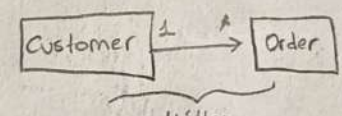


0, or more

1..* 1 or more

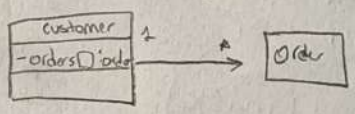
1 only 1

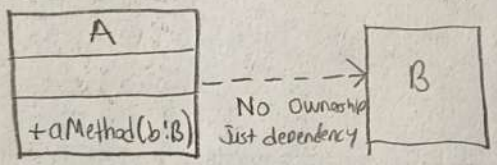
1..20 1 to 20



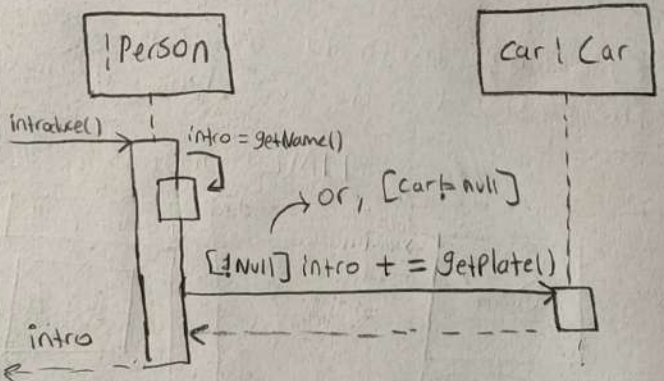
Hidden

OR





No Ownership Just dependency

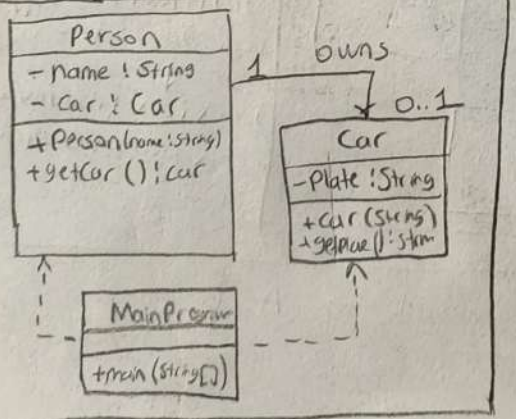


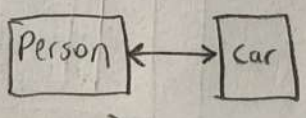
final olan Variable'ler set methodu ile değiştirilemez!

Constructor ile bir şeklik değer atanır.

değer hiç atanmazsa Compile hata verir.

Package



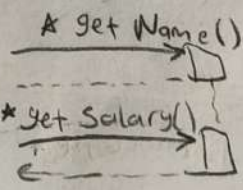


Two way if; "Car" class can send message to "Person" class.

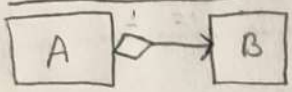
Add or remove from Array or List shown as;

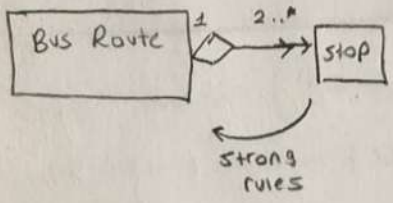
<<add Object>>

Loops shown As



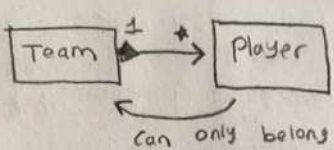
Aggregation





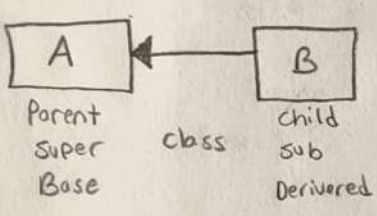
★ String compareTO
↓
if == 0, then = same

Composition



Inheritance

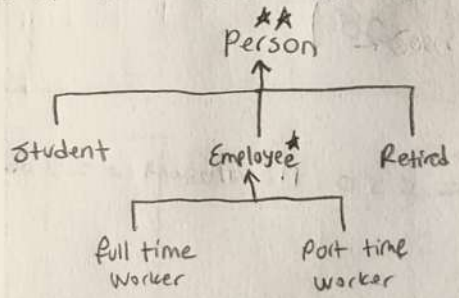
⇒ Creating new classes from an existing class. (Parent → children)
Genetic

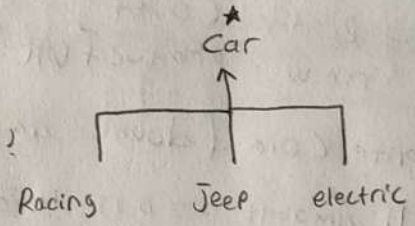


★ All fields and methods transferred to child class.
- but cant access private things.
+ protective can be accessed
↳ cant access but can Overriden (except final)
↳ can be used if overridden

Rules

Subclass have to take everything from super class.
↳ inherited methods can be changed with overriding. final methods cant be overriden.
↳ New members can added to subclass
↳ if you change super class it will effect subclass.

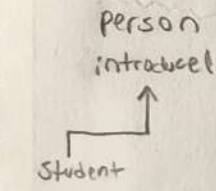




★ Java.lang.Object is super class of all classes.
inc. toString()
↓
We override it.

↳ Child class is custom version of super class.

Overriding



★ super can only be used once
★ super must be #1 on constructor

→ Public class Employee extends Person {

Public Employee (String name, int salary) {
★ Person ← super (name); → Person's constructor ★

Public class Manager extends Employee {

Public Manager (String name, int salary) {
★ Employee ← super (name, salary)
★ Public int getSalary() { → ★ Can use super's methods ★
return super.getSalary;

Aggregation (Has a)

↳ insan → kitap

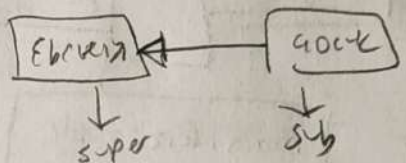
sahip ona onun varlığına bağlı değil

Inheritance (is a) student → Human → generalization

Composition (part of)

insan → el

varlol ona bağlı



Association → Bağlantı →

Dependency → - - - - ->

static → commonly shared between all instances of that class.

Member field → private int count; vs. methodların dışındaki variableler.

instantiated → c1 = new Book(); vs.

Overload → multiple methods, same name, different parameters

Override → super class'ın methodunu değişe

Private methods can overriden but cant accessed by subclass

---

final (public) methodlar override de yapılamaz.

final ile private zaten gereksiz.

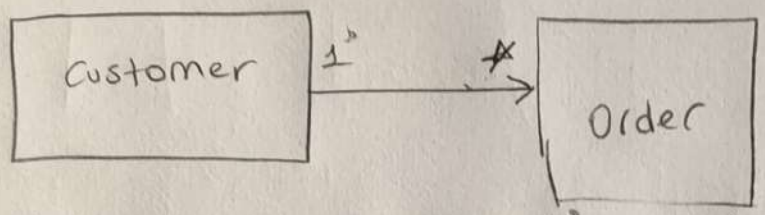
Overriding - polymorphism farkı!

Üst sınıftaki methodu değiştirme

Aynı method ismini kullanma

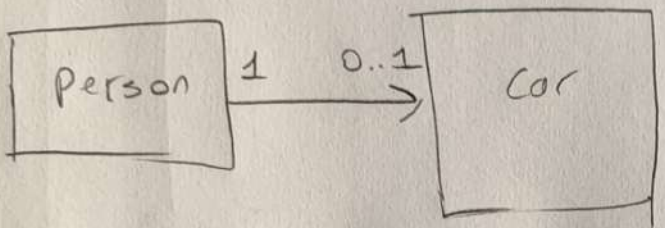
overriding bir polymorphism türü

daha genel overload da ona dahildir.

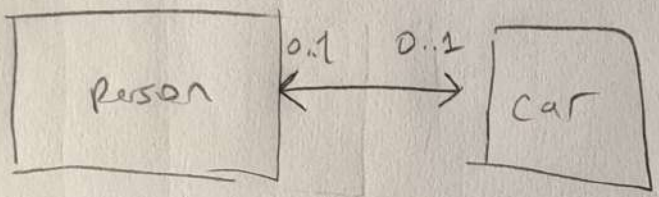


→ Assosiation (Bağlantı)

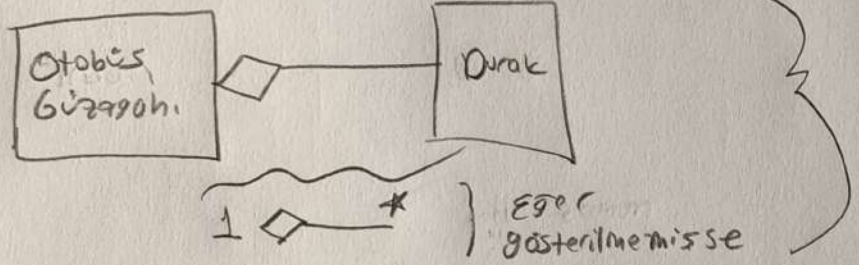
→ Her Orderin 1 Customer ile bağlantısı vardır
→ Customerin birden çok Orderi olabilir. (veya 0) (bağlantı)



→ Her insanın Arabası olabilir / olmayabilir (bağlantı)
→ Her Arabanın 1 person ile bağlantısı vardır.

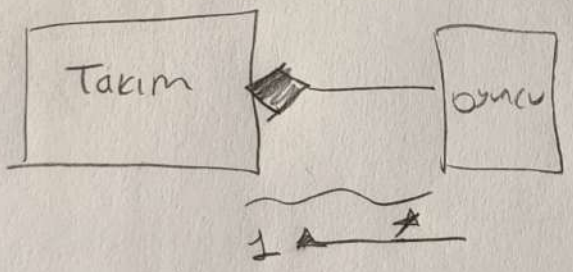


→ Her insanın Arabası var veya yok (bağlantı)
→ Her Arabanın person ile bağlantısı var veya yok



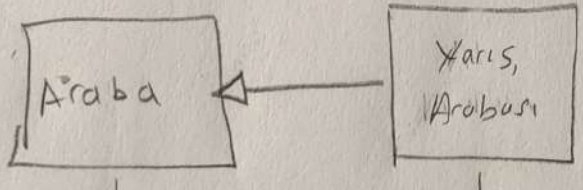
Varlığı Bağımsız = Aggregation Sahiplik

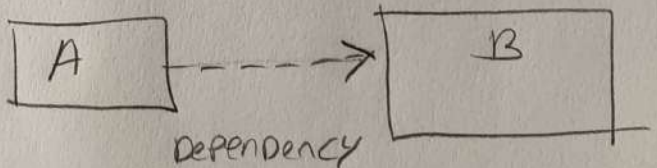
toplanma
küme
bir araya gelme.



Varlığı Bağımlı = Composition Sahiplik

Bileşim
Bitişik





Sahibi değilsin Ama kullanabilirsin
ör. B'nin metotlarını kullanma
A B'ye mesaj atıyor ama B sadece geçiçi atabiliyor.