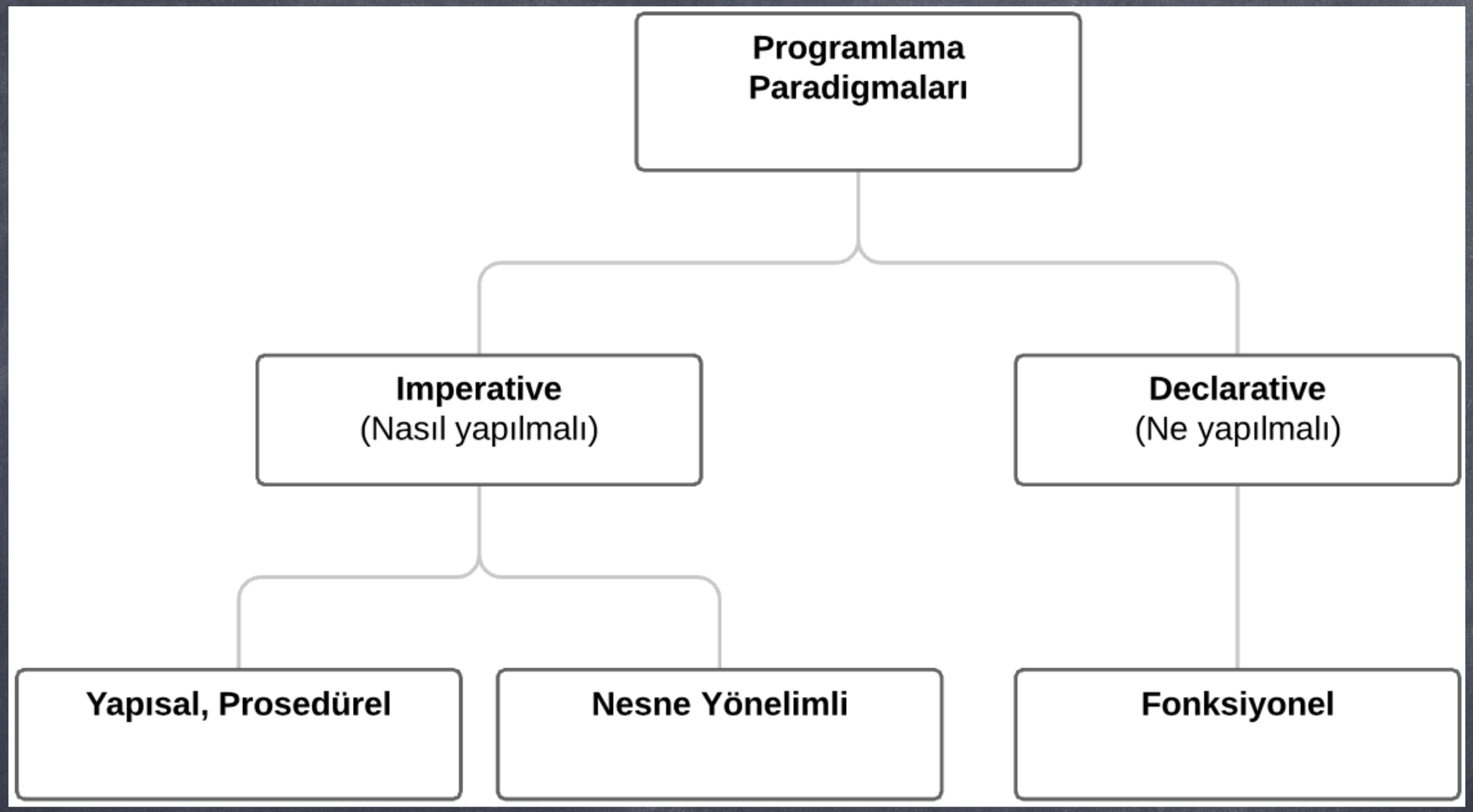
Nesne Yönelimli Analiz ve Tasarım

Programlama Paradigmaları

Programlama Paradigmaları



Programlama Paradigmaları

Paradigm	Description	Main traits	Related paradigm(s)	Critique	Examples
Imperative	Programs as statements that *directly* change computed state (datafields)	Direct assignments, common data structures, global variables		Edsger W. Dijkstra, Michael A. Jackson	C, C++, Java, Kotlin, PHP, Python, Ruby, Wolfram Language
Structured	A style of imperative programming with more logical program structure	Structograms, indentation, no or limited use of goto statements	Imperative		C, C++, Java, Kotlin, Pascal, PHP, Python, Wolfram Language
Procedural	Derived from structured programming, based on the concept of modular programming or the *procedure call*	Local variables, sequence, selection, iteration, and modularization	Structured, imperative		C, C++, Lisp, PHP, Python, Wolfram Language
Functional	Treats computation as the evaluation of mathematical functions avoiding state and mutable data	Lambda calculus, compositionality, formula, recursion, referential transparency, no side effects	Declarative		C++,[1] Clojure, Coffeescript,[2] Elixir, Erlang, F#, Haskell, Java (since version 8), Kotlin, Lisp, Python, R,[3] Ruby, Scala, SequenceL, Standard ML, JavaScript, Elm, Wolfram Language
Event-driven including time-driven	Control flow is determined mainly by events, such as mouse clicks or interrupts including timer	Main loop, event handlers, asynchronous processes	Procedural, dataflow		JavaScript, ActionScript, Visual Basic, Elm
Object-oriented	Treats datafields as *objects* manipulated through predefined methods only	Objects, methods, message passing, information hiding, data abstraction, encapsulation, polymorphism, inheritance, serialization-marshalling	Procedural	Wikipedia, others[4][5][6]	Common Lisp, C++, C#, Eiffel, Java, Kotlin, PHP, Python, Ruby, Scala, JavaScript[7][8]
Declarative	Defines program logic, but not detailed control flow	Fourth-generation languages, spreadsheets, report program generators			SQL, regular expressions, Prolog, OWL, SPARQL, XSLT

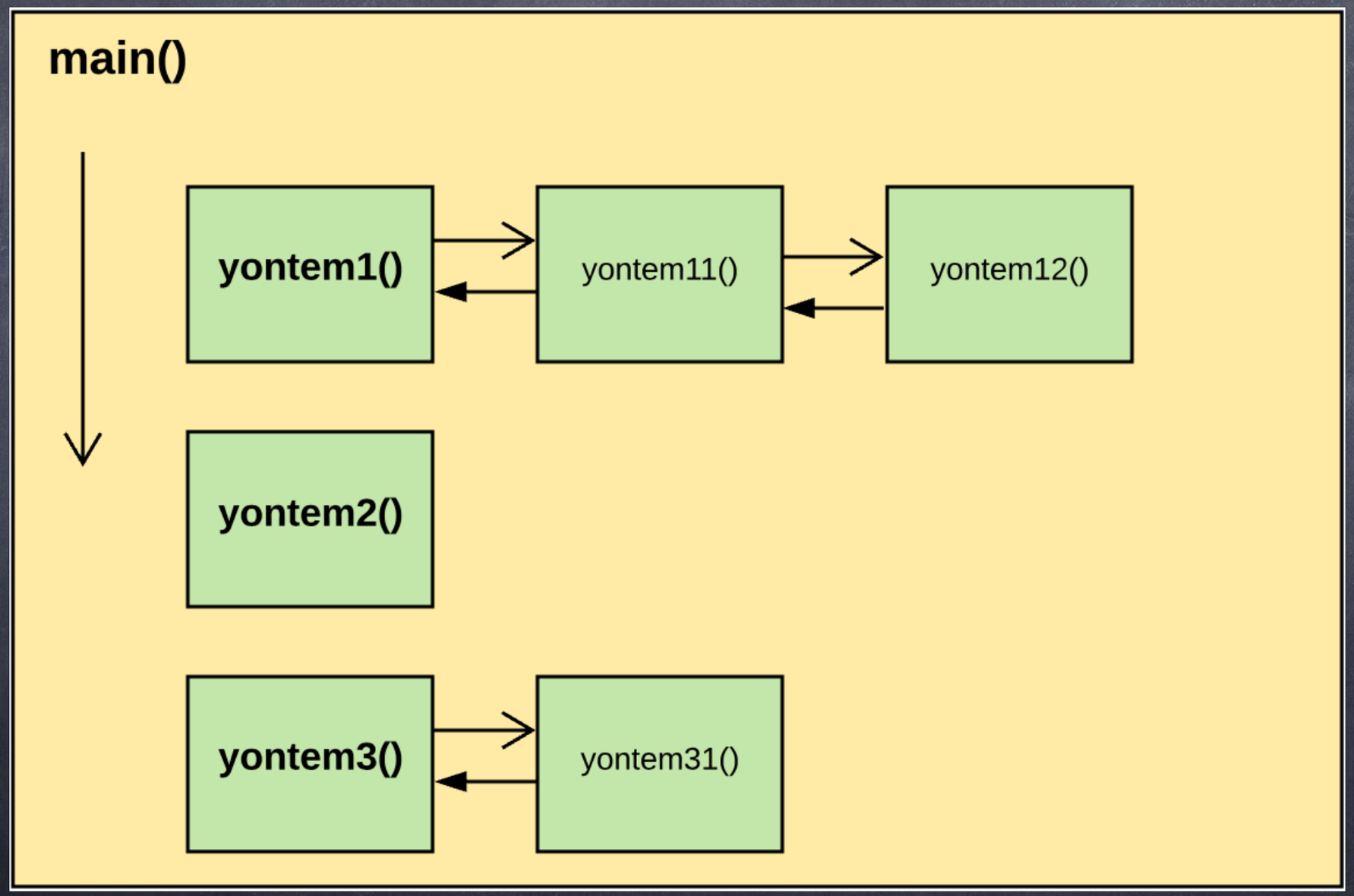
https://en.wikipedia.org/wiki/Structured_programming

Yapısal (Prosedürel) Programlama Paradigması

* Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures and for and while loops in contrast to using simple tests and jumps such as the goto statement which could lead to "spaghetti code" which is difficult both to follow and to maintain.

https://en.wikipedia.org/wiki/Structured_programming

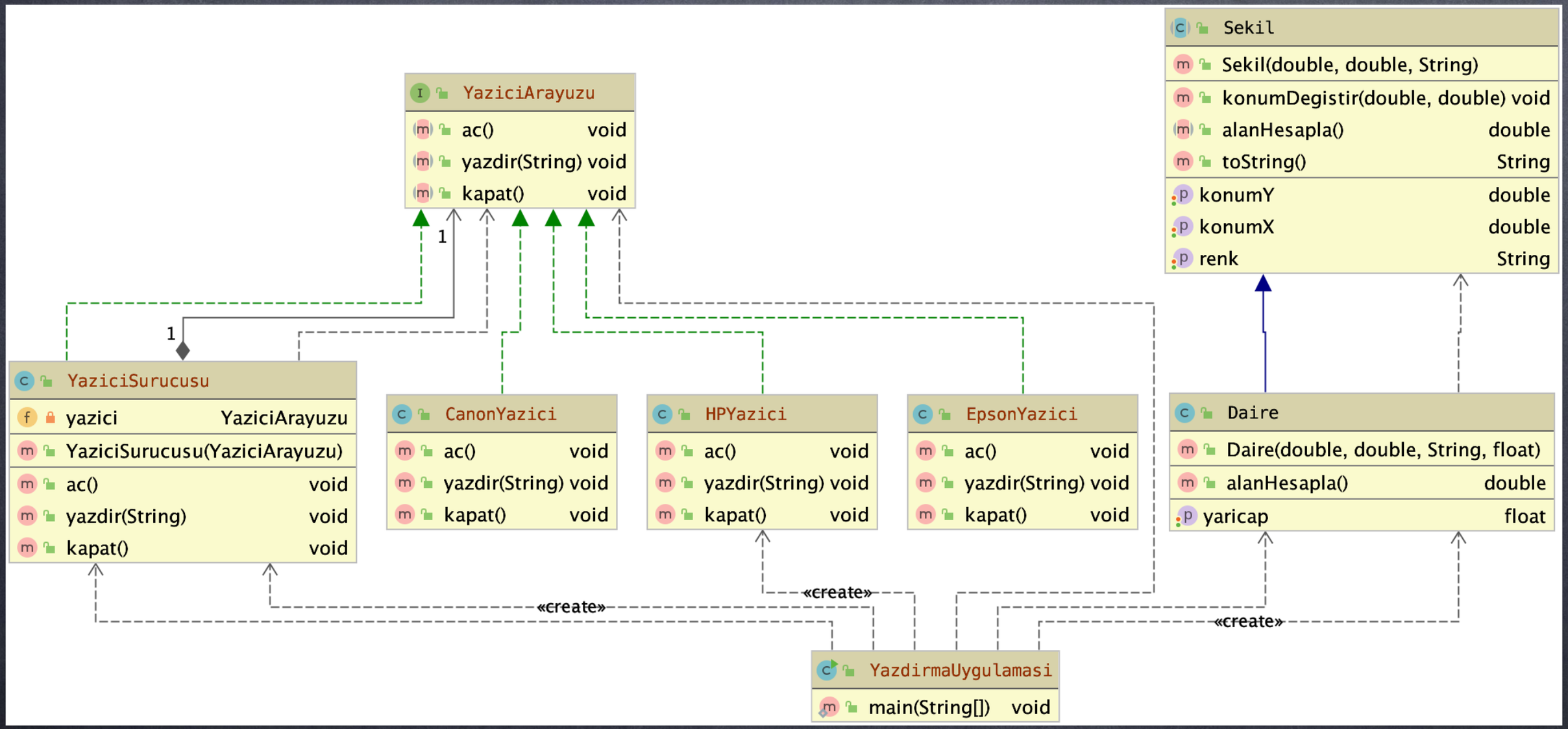
Yapısal (Prosedürel) Programlama Paradigması



Nesne Yönelimli Yazılım Geliştirme Paradigması

* Yapısal teknikte doğrudan probleme odaklanılır ve problemin çözümüne ilişkin yöntemler geliştirilir (control-centric).
* Nesne yönelimli programlama tekniğinde ise temel bileşen nesnedir (data-centric) ve programlar nesnelerin birlikte çalışmasından meydana gelir.
* Nesne hem veriyi hem de bu veriyi işleyen yöntemleri içerir. Yazılım geliştiriciler dikkatlerini nesneleri oluşturan sınıfları geliştirmeye yoğunlaştırır.
* Yapısal teknikte bir fonksiyon herhangi bir görevi yerine getirmek için veriye ihtiyaç duyarsa, gerekli veri parametre olarak gönderilir. NYP de ise yerine getirilmesi gereken görev nesne tarafından icra edilir ve fonksiyonlar verilere parametre gönderimi yapılmaksızın erişebilirler.

Nesne Yönelimli Yazılım Geliştirme Paradigması

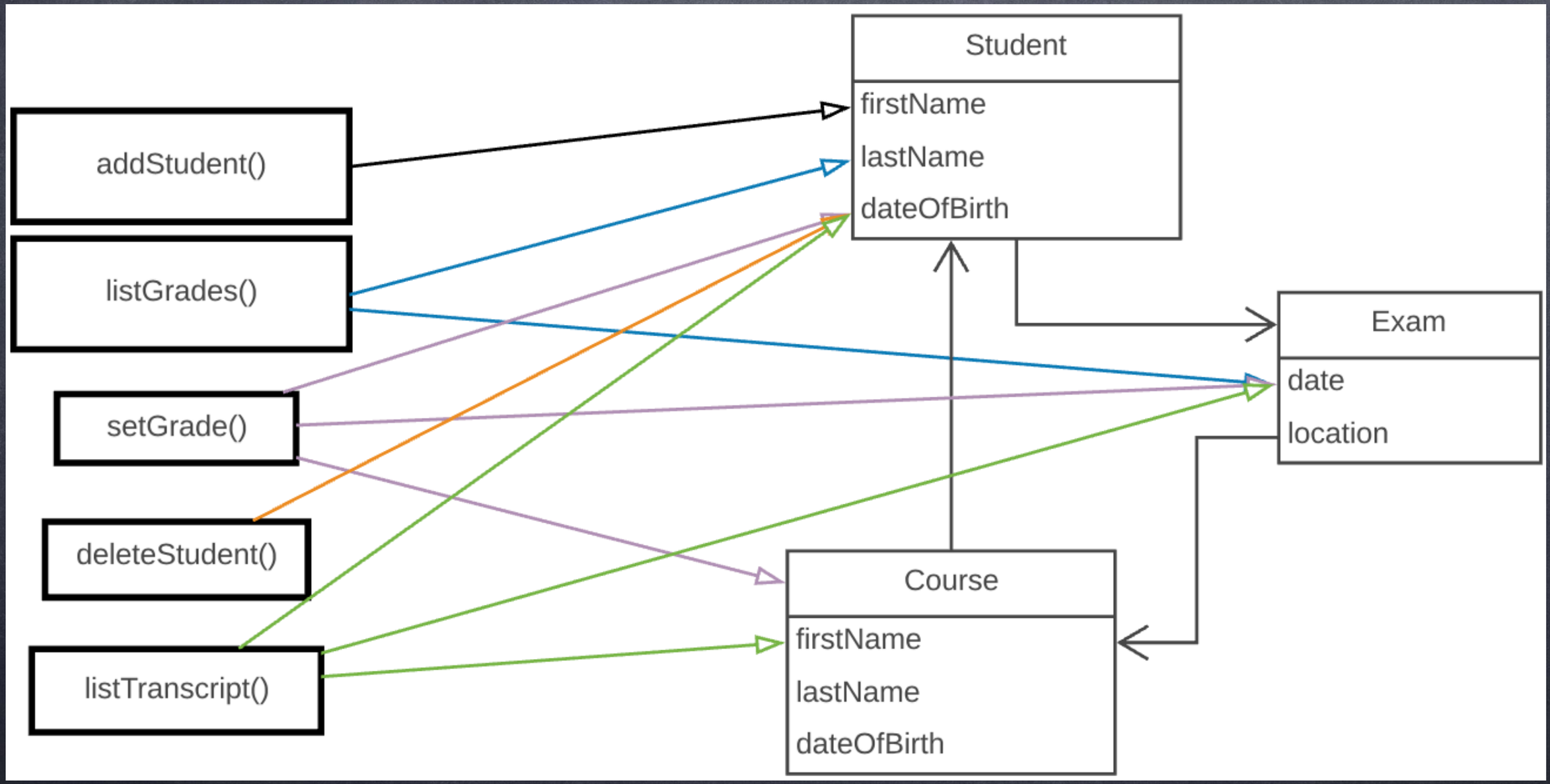


Nesne Yönelimli Yazılım Geliştirme Paradigması

* Sistem büyüdükçe ilişkiler/bağımlılık daha da karmaşıklaşır.
* Sonradan değişiklik zorlaşır.
* Program içerisinde değişiklik (ekleme, çıkarma, düzeltme) yapmak zorlaşır ve beklenmeyen etkilere neden olabilir...
* Örneğin;
  * Ogrenciler tablosunda ogrencinin dogum tarihi iki haneli
  * Bu alanı 4 haneli yapmak istiyoruz...
  * Bu veriyi kullanan yöntemlerinde güncellenmesi gerekecektir.
  * Hangi yöntemler hangi veriye erişiyor ?

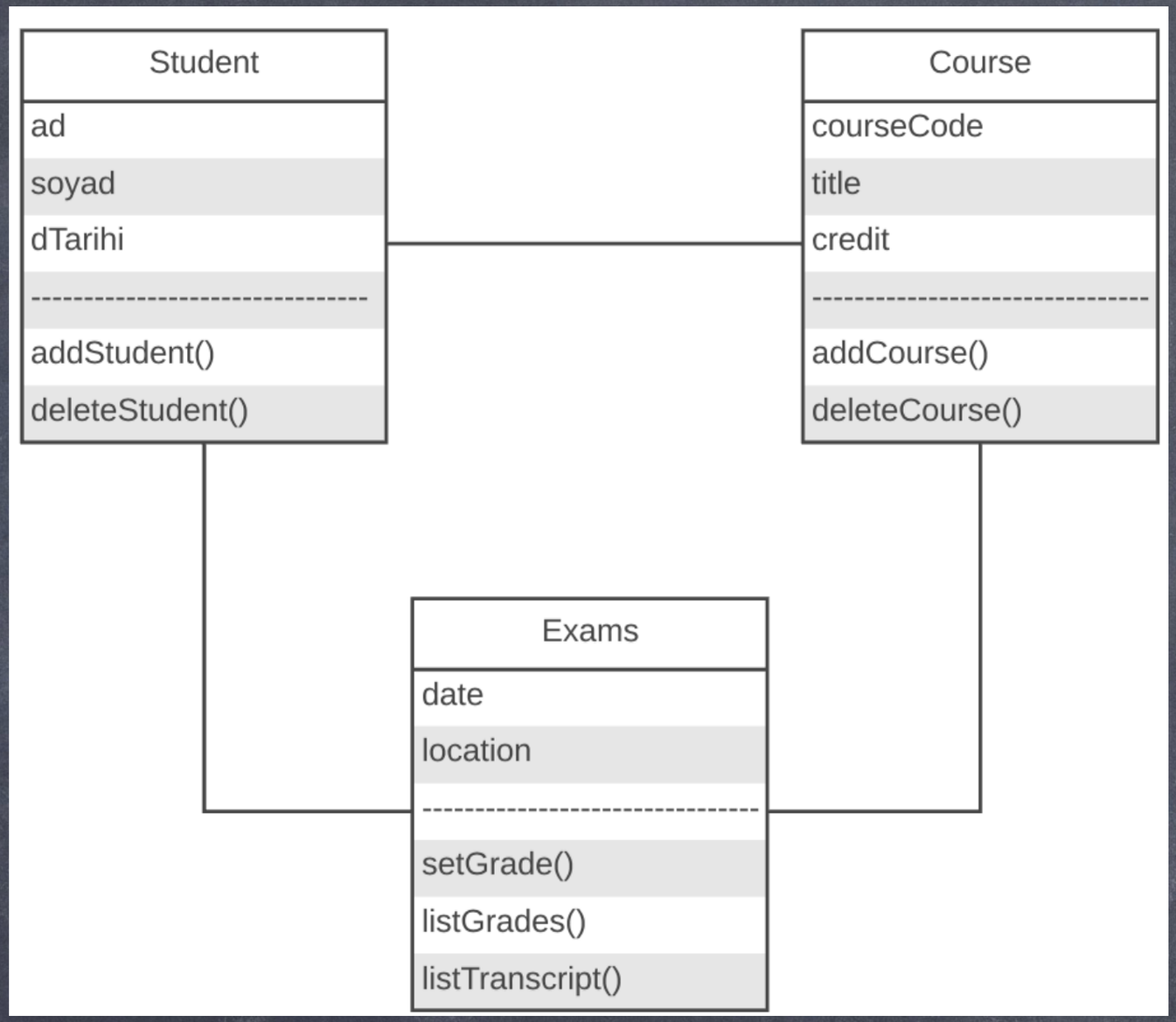
Nesne Yönelimli Yazılım Geliştirme Paradigması

Yöntemler **Veri**



Hangi yöntemler hangi veriye erişiyor ?

Nesne Yönelimli Yazılım Geliştirme Paradigması



High Coherence
Modüller tek ve özel bir işi, mükemmel bir şekilde yapmalı. Alt sistemler içerisindeki sınıflar benzer işi yapmalı ve ilgili olmalı. "Cohesion" Bunun ölçüsüdür .

Low Coupling
Aradaki bağıntılar ne kadar fazla olursa nesne içerisinde yapılacak köklü değişiklik ona bağlı olan modülleride etkileyecektir.

Nesne Yönelimli Yazılım Geliştirme Paradigmasının Temel Özellikleri

* Encapsulation (Information Hiding)
* Inheritance
* Polymorphism
* Abstraction
* Modular Programming
* Code Reuse
* Maintenance
* Design Principles (SOLID)
* Design Patterns

Uygulamalar
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders2

Fonksiyonel Programlama

* (Pure) saf fonksiyonların birleşiminden oluşur
  * aynı giriş için aynı çıkış üretilir
  * paylaşılan değişken yoktur
  * global değişken yoktur
  * parametre olarak gelen nesnenin durumları değiştirilemez.
* Tekrarlı yapılar (döngü) yoktur.

```
List<Kitap> kitaplarV = new Vector<Kitap>();
kitaplarV.add(new Kitap("Vektör Veritabanı ",100.00));
kitaplarV.add(new Kitap("Vektör Nesne Yönelimli ",125.00));
kitaplarV.add(2,new Kitap("V Bilgisayar Ağları",150.00));

System.out.println(kitaplarV);

for (int i=0; i<kitaplarV.size();i++)
    System.out.println(kitaplarV.get(i).getAdi());

//for each ile erişim
for(Kitap kitap:kitaplarV)
    System.out.println(kitap.getAdi());

System.out.println("************Lambda Expression************'

kitaplarV.forEach(kitap -> System.out.println(kitap));

System.out.println("************Lambda Expression 2**********

kitaplarV.forEach(kitap -> System.out.println(kitap.getAdi()));
```

https://andriybuday.com/2010/02/lambda-expression-or-simple-loop.html