Decorator Örüntüsü

Soyut Tiplere Bağımlılık

Her zaman abstract type'lara veya interface'lere depend et, **concrete type'lara depend etme**meye çalış
(**Dependency Inversion Principle**)

Inheritance Yerine Composition

Mümkün olduğunca **inheritance yerine composition'ı ve delegation**'ı tercih et

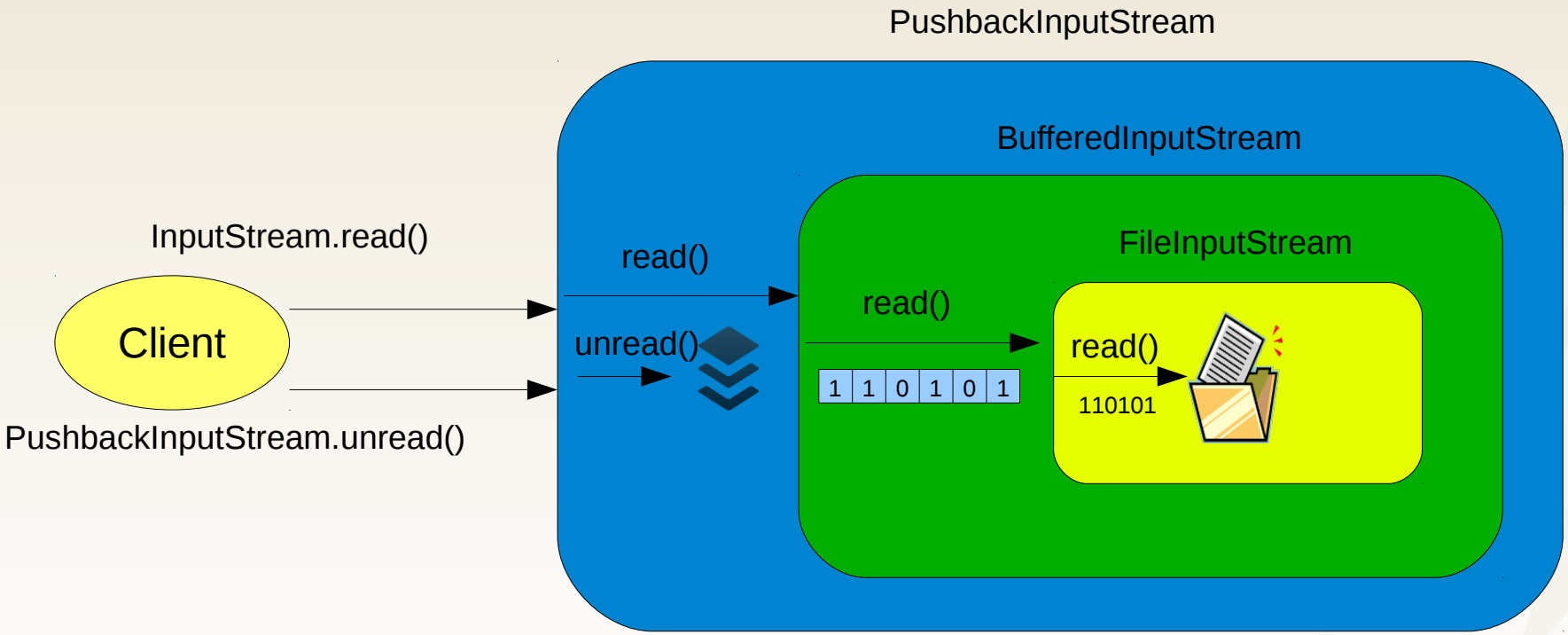
Decorator ile İlave Davranışlar Eklenmesi

- Zaman zaman nesneye sınıfın sağladığı kabiliyetin dışında, **ilave başka bir davranış kazandırmak** gerekebilir
- Bu ilave davranışı **kalıtım kullanmadan,** bir takım tercihlere veya durumlara göre nesneye **dinamik olarak** kazandırmak, mevcut kodu etkilememek açısından en doğrusudur
- Decorator örüntüsü ile **nesne düzeyinde** davranışların çeşitlendirilmesi mümkündür

Decorator ile İlave Davranışlar Eklenmesi

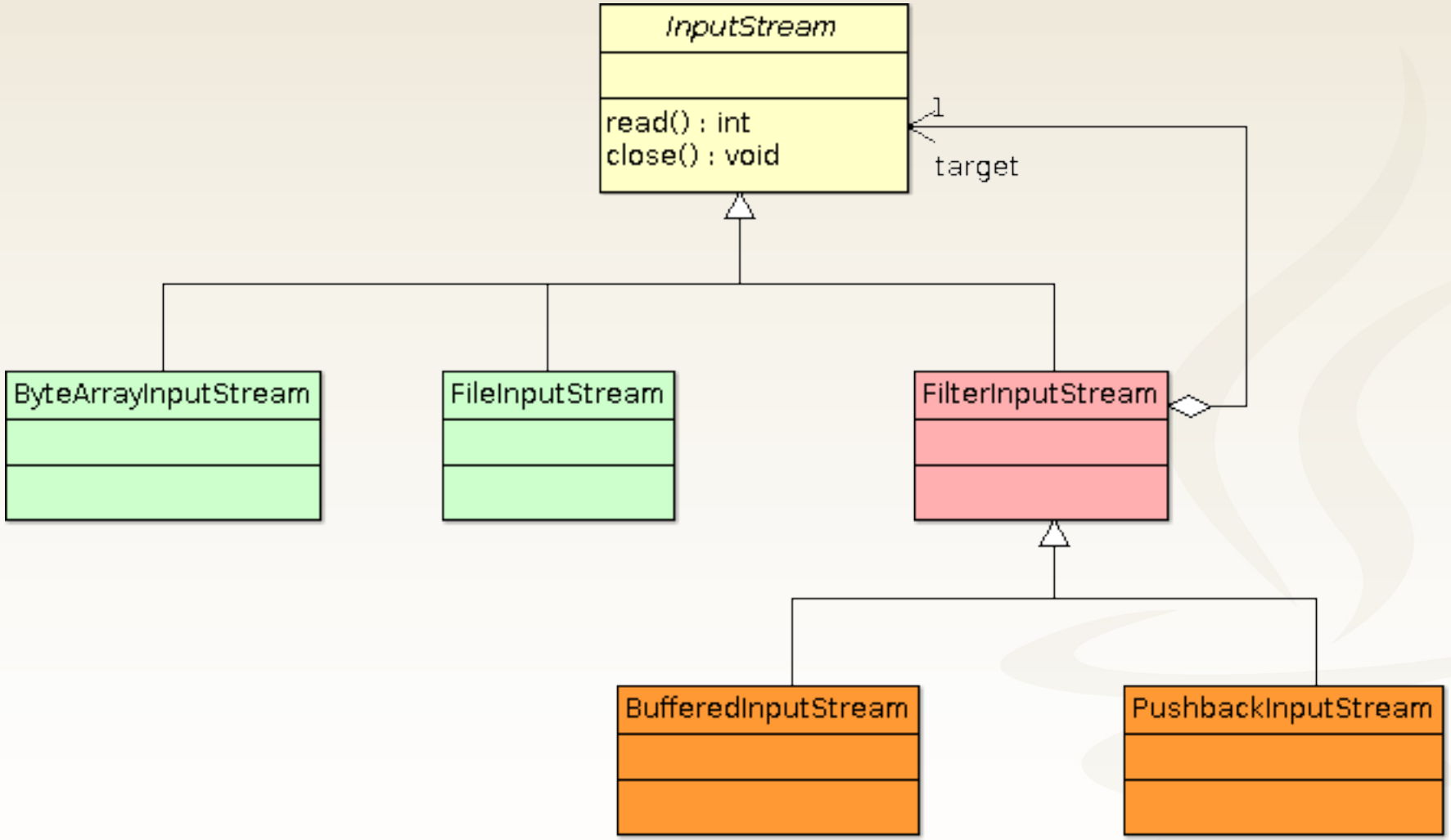
- Ancak **ilave kabiliyeti** kazandırmak için ilk akla gelen çoğunlukla **kalıtım** kullanmak olur
- Kalıtım ile her bir yeni özelliğin **mevcut diğer özelliklerle kombinasyon**a girebileceği düşünülürse, bunun sonucunun “sınıf sayısında geometrik oranda artış” olacağı aşikardır
- İlave kabiliyet kalıtım yerine **composition** ile kazandırılabilirse hem **mevcut kod üzerinde değişiklik yapılmamış** hem de **sınıf sayısı geometrik oranda artmamış** olur

Decorator ve Java I/O API

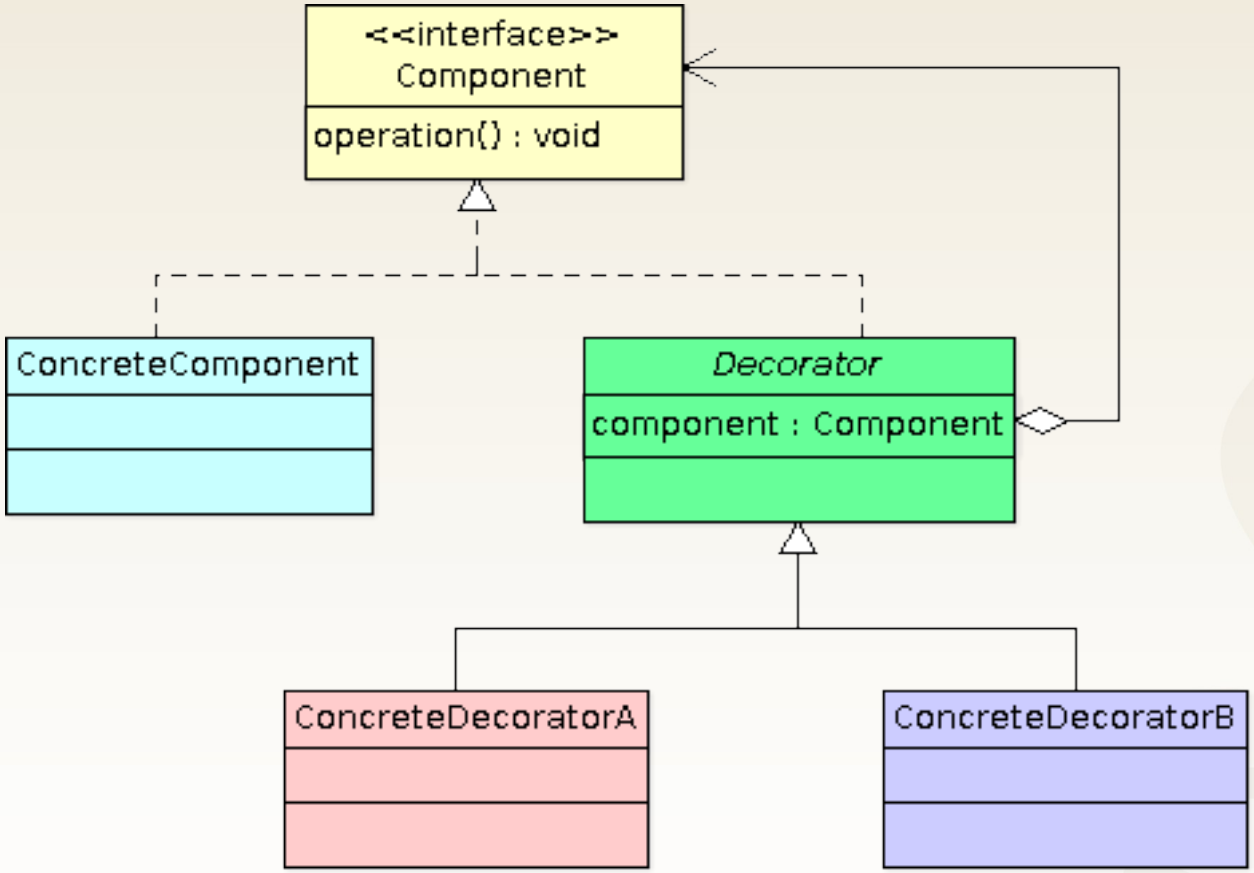



```
InputStream is = new PushbackInputStream(
                     new BufferedInputStream(
                         new FileInputStream ("/myfile.txt")));

is.read();

((PushbackInputStream)is).unread();
```

Decorator ve Java I/O API



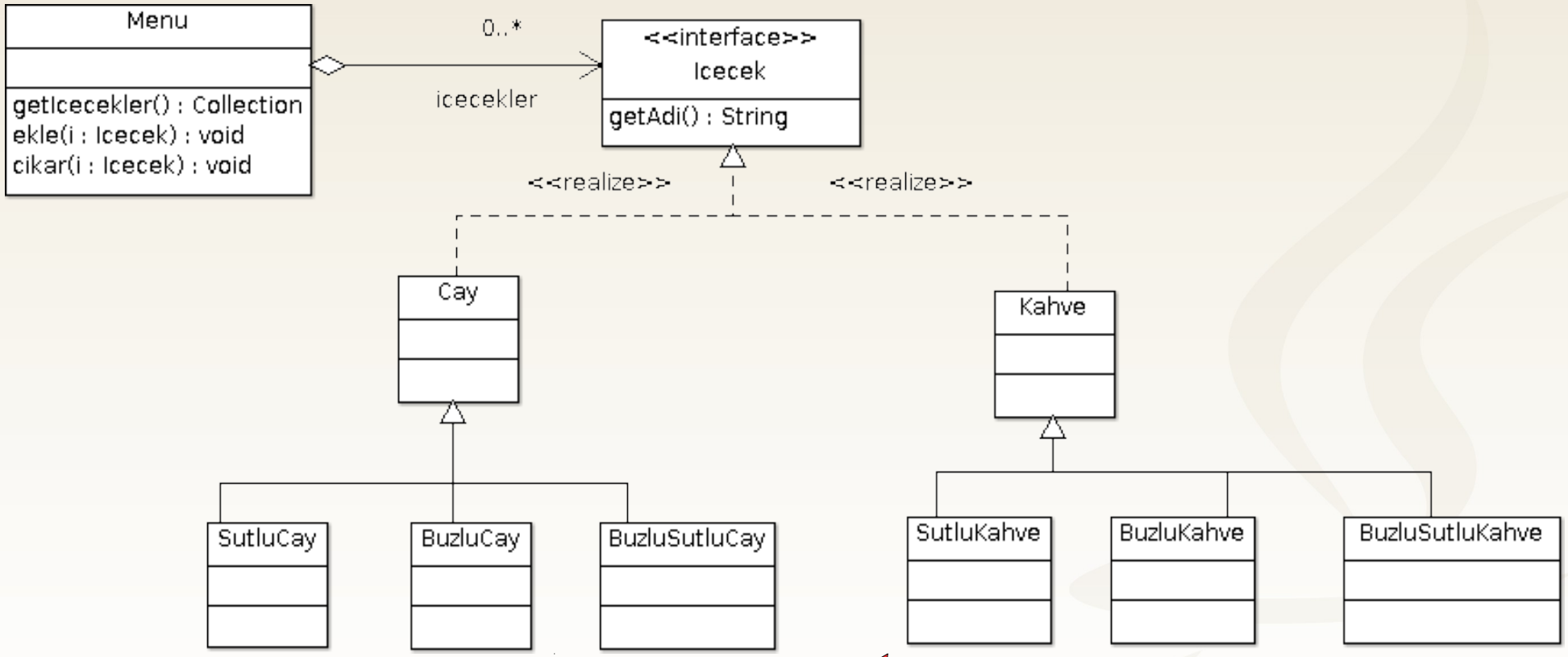
Decorator Sınıf Diagramı



LAB ÇALIŞMASI: Decorator

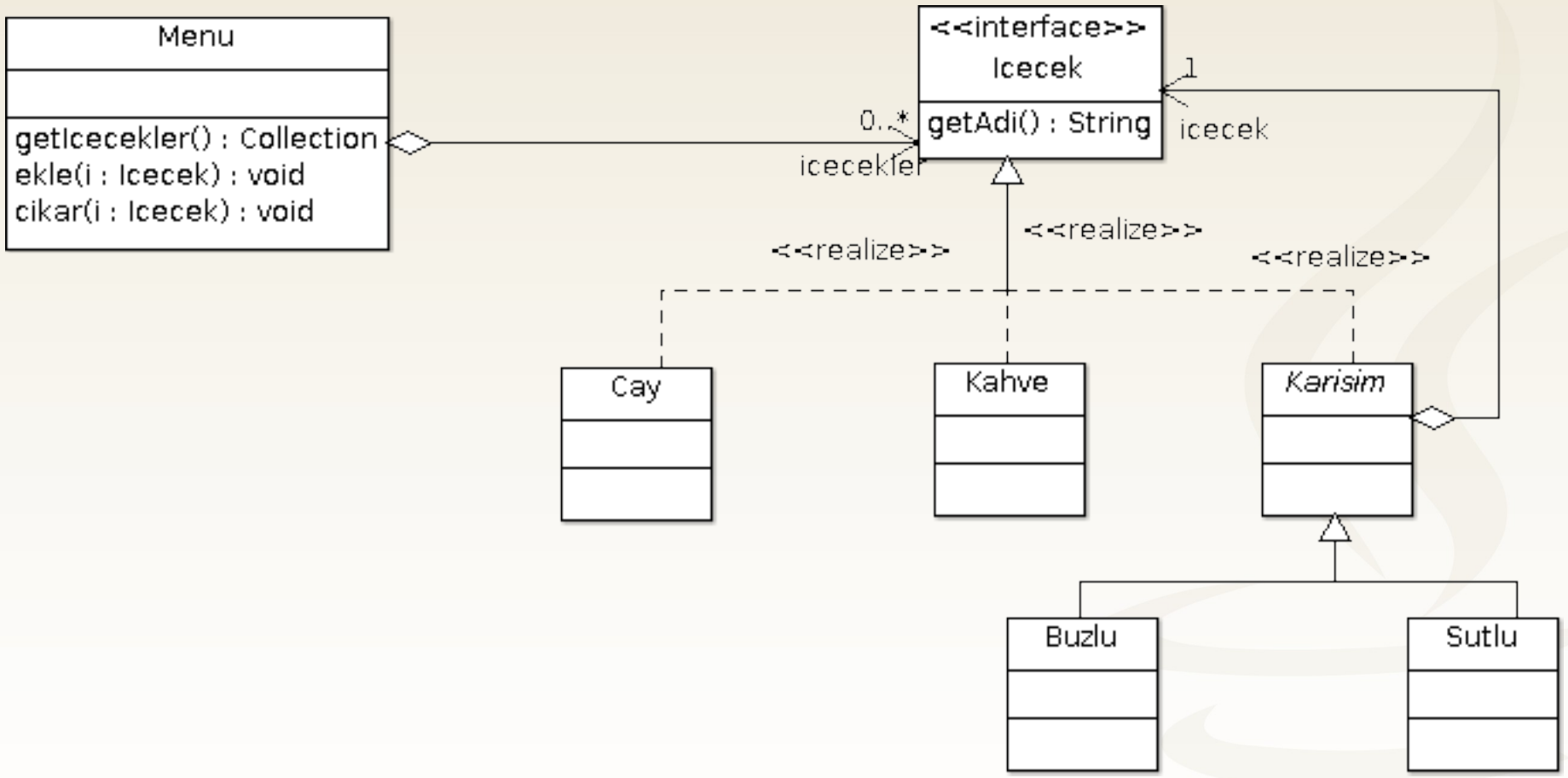
- Yeni açılan bir kafenin menüsünün içecekler kısmında çay ve kahve şeklinde iki farklı içecek seçeneği mevcuttur
- Müşteriler çay ve kahve siparişlerini sade verebildikleri gibi, sütlü veya buzlu olarak da verebilirler
- Tercihler bu karışımların birbirleri ile kombinasyonu şeklinde de olabilmektedir. Örneğin buzlu ve sütlü kahve siparişi gibi.
- Menüde yer alan içeceklerin yukarıda belirtilen tür ve karışımlarda olmasını sağlayan ve sınıf patlamasına yol açmayan bir tasarım yapılması istenmektedir

LAB ÇALIŞMASI: Decorator



Sınıf patlaması!

LAB ÇALIŞMASI: Decorator



Inheritance Yerine Collaboration ve Responsibility

Taksonomik **sınıflandırma yaklaşımı** yerine **davranış ve sorumluluklar**a daha çok odaklan!

Decorator Örüntüsünün Sonuçları

- Yeni özellikler eklenirken sistemdeki sınıf sayısının **geometrik olarak artmasın**ın önüne geçilir
- Bir sınıfı tasarlarken sınıfın sahip olması gereken **ana sorumluluk ile** bu davranışın üzerine eklenebilecek **ilave kabiliyetler**in daha net ayrımına varılabilir
- Birden fazla kabiliyetin **bir sınıf içerisinde birikmesi**nin önüne geçmeye yardımcı olur

İletişim

www.harezmi.com.tr

www.java-egitimleri.com

info@harezmi.com.tr

info@java-egitimleri.com

@HarezmiBilisim

@JavaEgitimleri