Nesne tabanlı programlama (NTP) özellikleri

- Çokbiçimlilik - polimorphizm
- Miras – inheritance
- Soyutlama – abstraction
- Kapsülleme – encapsulation

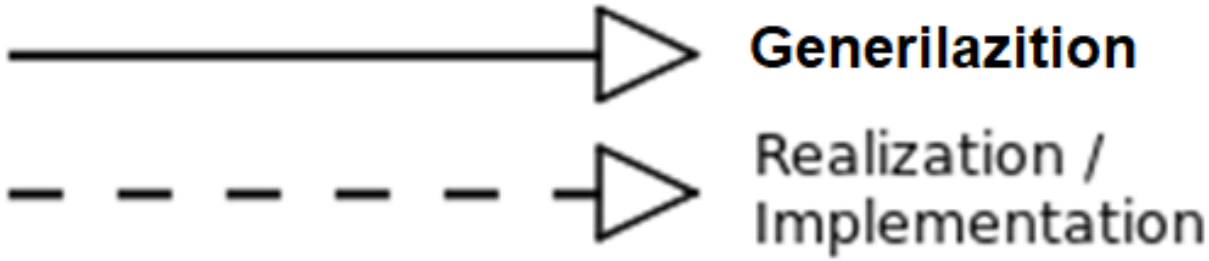
Miras – inheritance

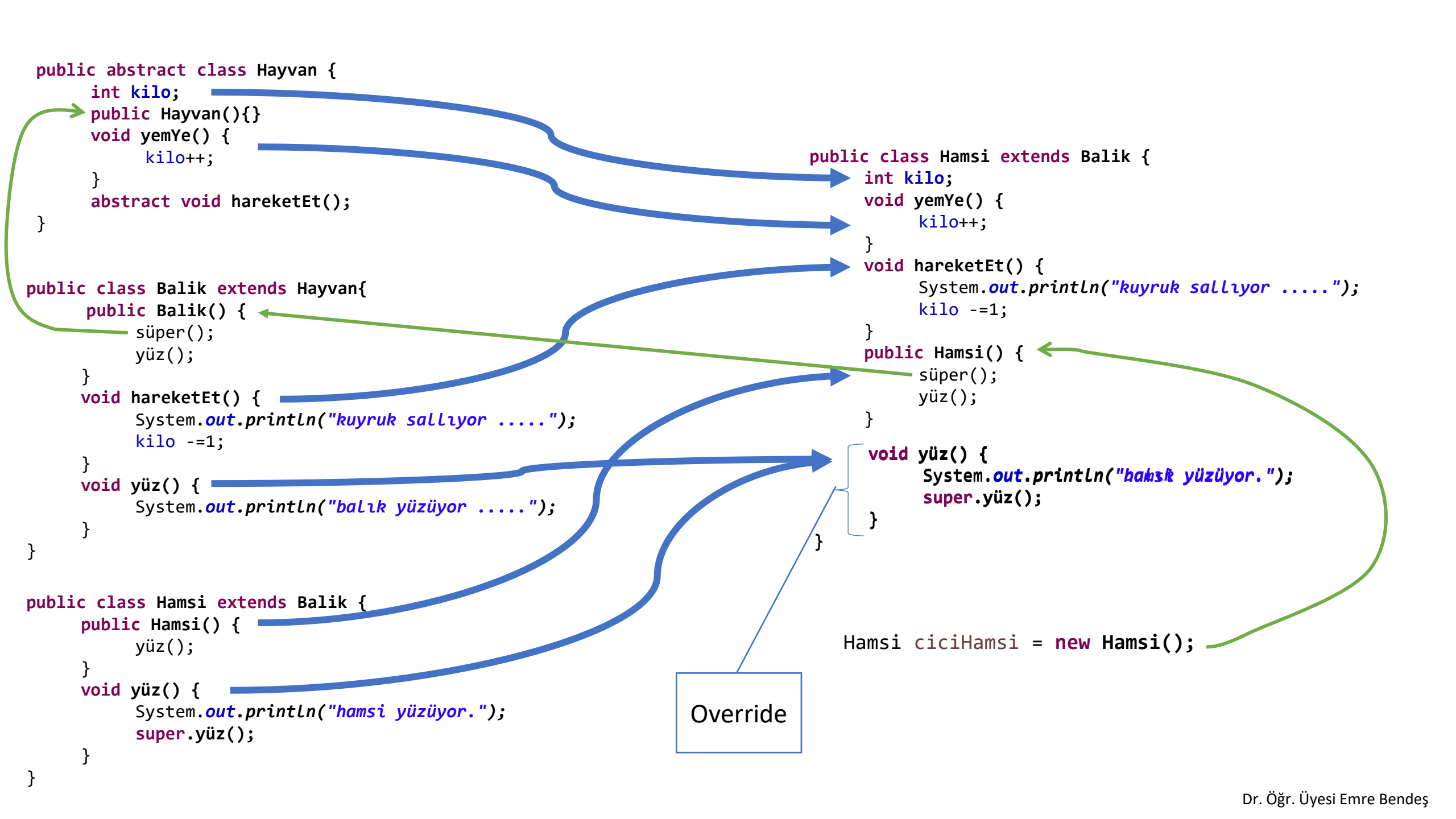
Bir sınıfın miras aldığı sınıfın özelliklerini ve metotlarını üzerinde toplaması

Var olan veri türlerini kullanarak yeni bir veri türü üretmek

Miras – inheritance

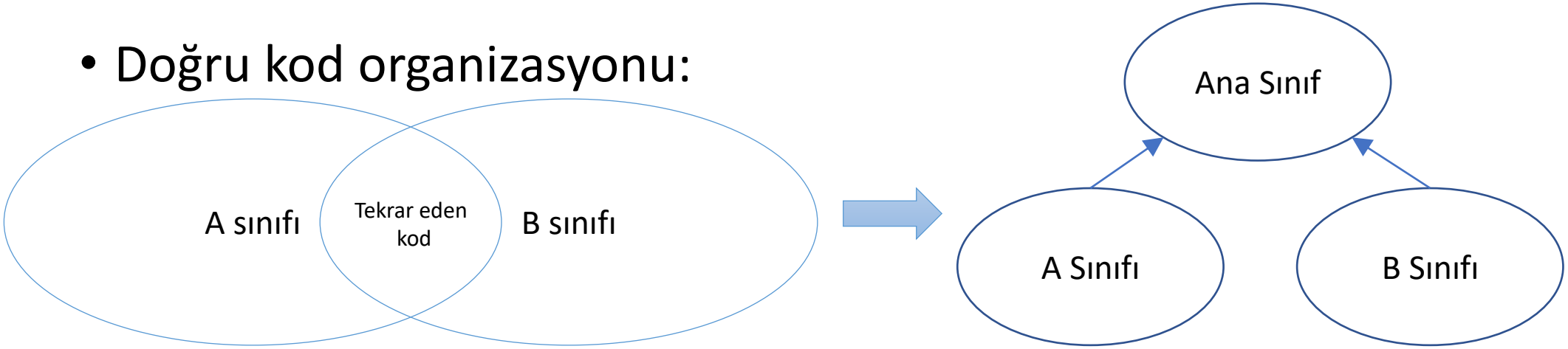
- Ana sınıf – alt sınıf (base - child)
- Sınıflar arası **is-a** (generalization) ilişkisi kurar
- Sınıfların özelliklerini alt sınıflara bırakması
- Temel özellikler ana sınıfta toplanır
- Alt sınıflar temel özelliklere kendi özelliklerini eklerler
- Daha iyi organize edilmiş kod sağlar.
- Hiyerarşik tasarım sağlar
- Soyutlamanın önünü açar.




public abstract class Hayvan {
int kilo;
public Hayvan(){}
void yemYe() {
kilo++;
}
abstract void hareketEt();
}
public class Balik extends Hayvan{
public Balik() {
süper();
yüz();
}
void hareketEt() {
System.out.println("kuyruk sallıyor .....");
kilo -=1;
}
void yüz() {
System.out.println("balık yüzüyor .....");
}
}
public class Hamsi extends Balik {
public Hamsi() {
yüz();
}
void yüz() {
System.out.println("hamsi yüzüyor.");
super.yüz();
}
}
public class Hamsi extends Balik {
int kilo;
void yemYe() {
kilo++;
}
void hareketEt() {
System.out.println("kuyruk sallıyor .....");
kilo -=1;
}
public Hamsi() {
süper();
yüz();
}
void yüz() {
System.out.println("hamsi yüzüyor.");
super.yüz();
}
}
Hamsi ciciHamsi = new Hamsi();
Override

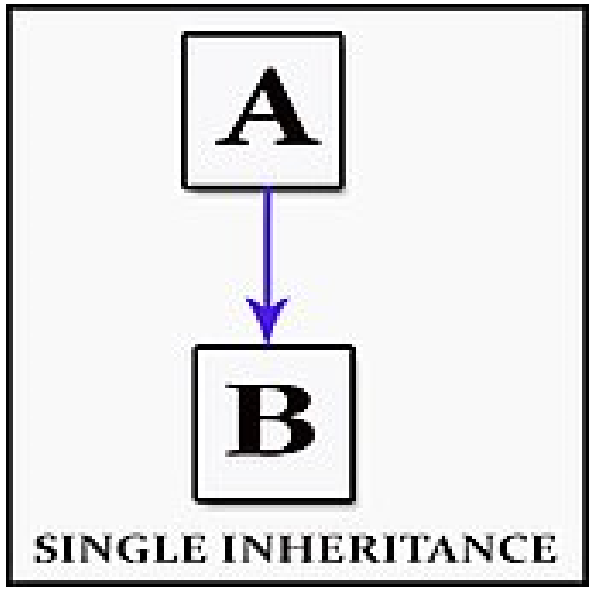
Neden Miras Kullanalım

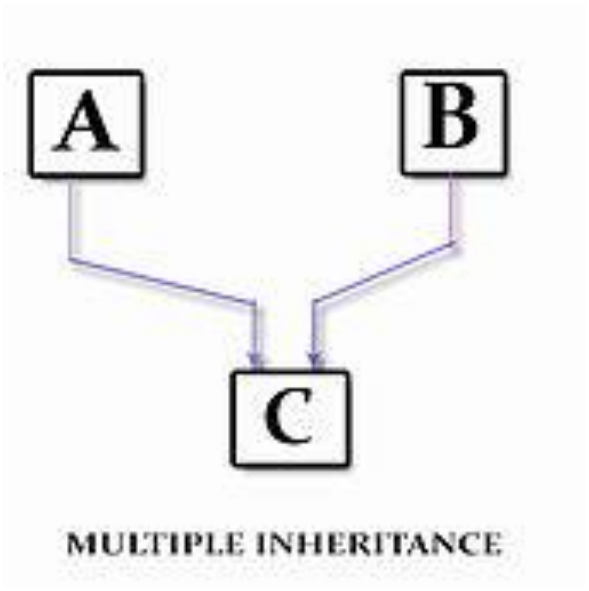
- Doğru kod organizasyonu:

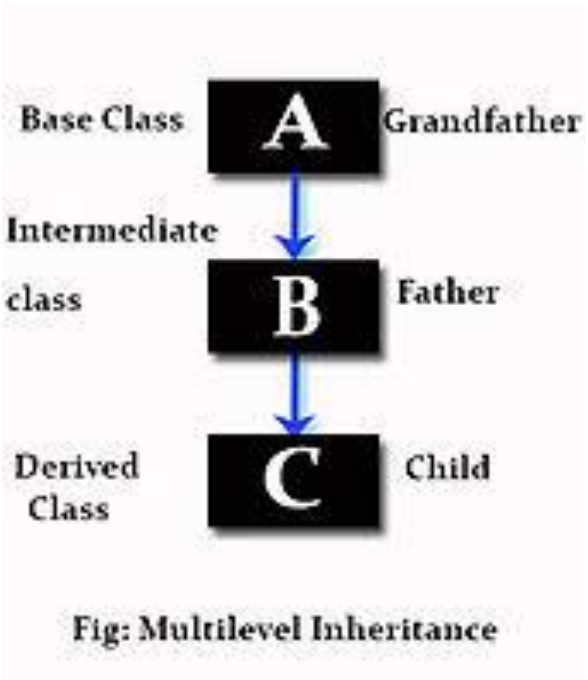


- Nesneler hiyerarşisi oluşturma
- Çok biçimlilik Önünü Açar
  - Downsampling

Miras Çeşitleri







Görüntü kaynağı: https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)

Kapsülleme – encapsulation

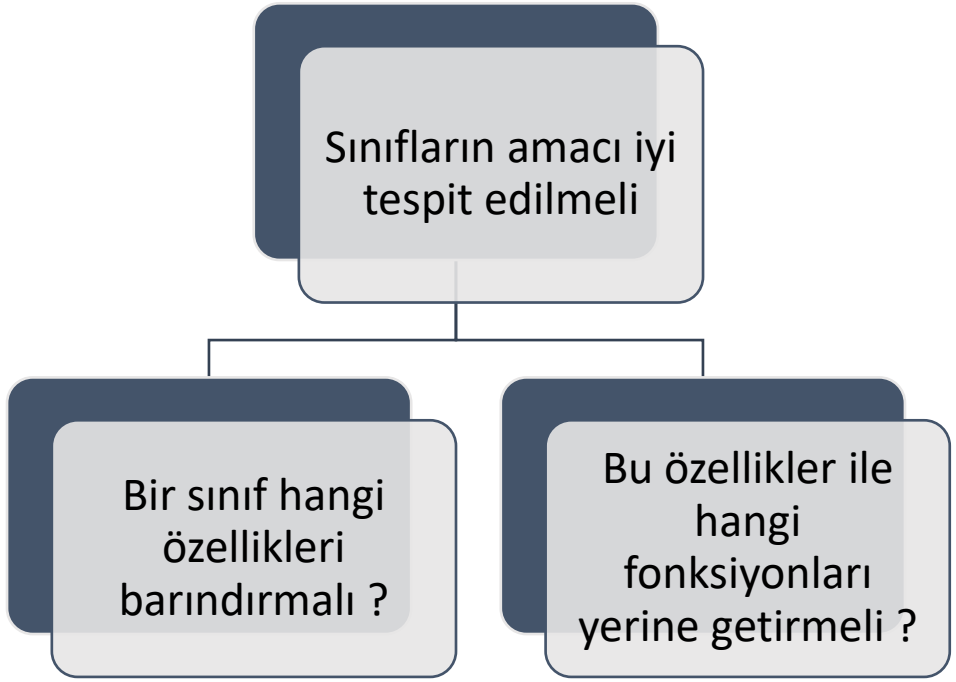
Sınıf iç yapısını korumak

Dr. Öğr. Üyesi Emre Bendeş

Kapsülleme – encapsulation

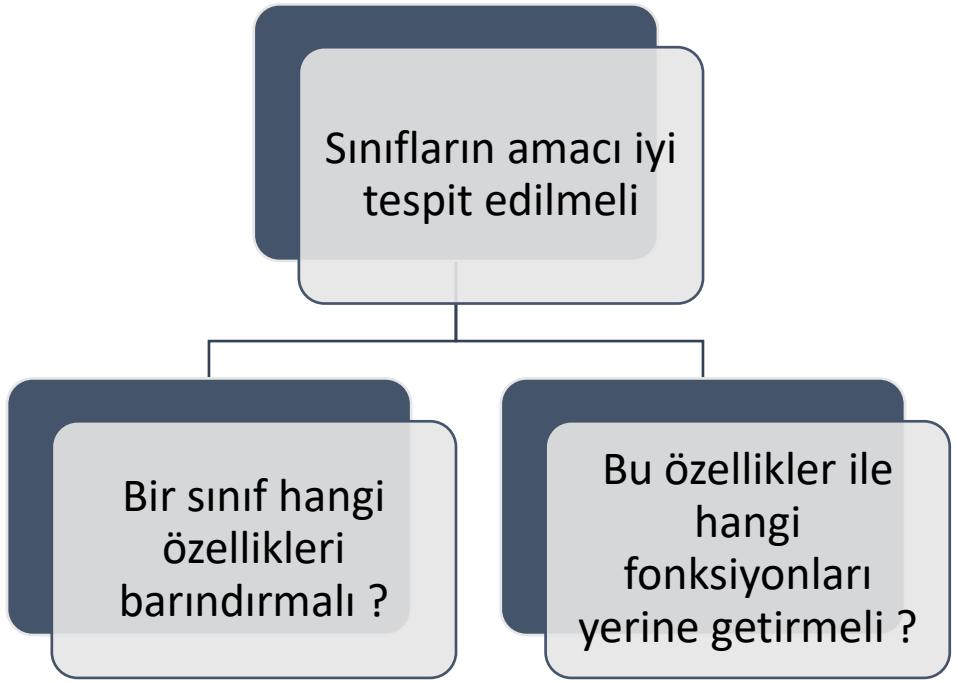
- Sınıfın iç yapısına müdahaleyi düzenler
- Sınıfı tasarımı açısından düşünülmeli
- Erişim belirleyiciler (modifiers)
  - Public
  - Private
  - Protected
  - Default
- Getter ve Setter metodları
  - Sınıf özelliklerine erişimin kontrollü olmasını sağlar

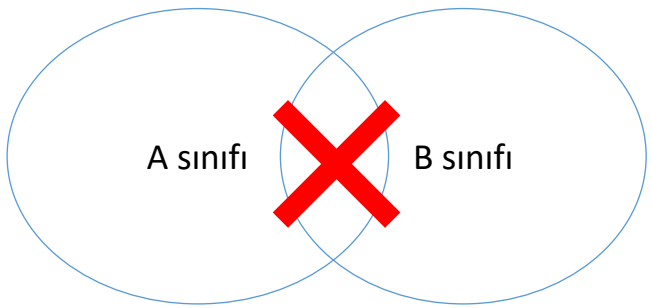
Kapsülleme – encapsulation

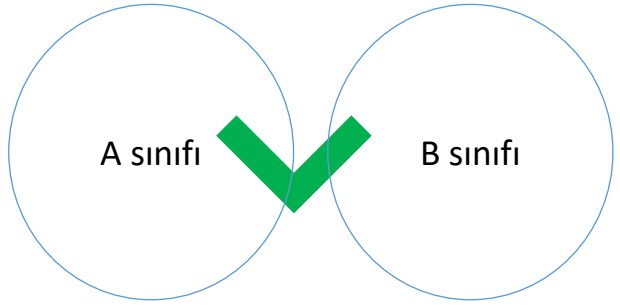


- Her sınıfın bir amacı olmalı
- Amaç ile ilgili bütün işlemler bu sınıfta gerçekleştirilmeli
- İstense dahi işlemleri yapan metotlar dışarıdan erişime kapalı olmalı

Kapsülleme – encapsulation







Kapsülleme ne sağlar ?

- Kod karmaşıklığı ortadan kaldırılmış olur (spagetti kod)
- Modülariteye katkısı olur
- Doğru tasarım sağlar

Soyutlama – abstraction

Nasıl çalıştığınla ilgilenmiyorum, bana sonuç ver

Dr. Öğr. Üyesi Emre Bendeş

Soyutlama – abstraction

Tanım:

- Soyutlama, yalnızca ilgili ayrıntıları belirleme ve gerekli olmayan ayrıntıları göz ardı etme sürecidir.
- Farklı kod parçalarının kompleks kısımlarını, sahip oldukları ortak davranışlar, amaçlar, karakteristik özellikler arkasında saklamak sayesinde daha anlaşılır ve kolay kullanılabilir kod yazmaktır.

Ana amaç

- Karmaşıklığı ele almak
- Gereksiz detayları kullanıcıdan gizlemek
- Bağımlılıkları azaltmak

Sonuç:

- Kullanıcı alt planda yatan detayları bilmeden farklı problemlere odaklanabilir.

Gerçek hayattan soyutlama örnekleri

Bir kahve makinasını nasıl çalıştığı ile ilgilenilmez, sadece kullanılır.

Araç kullanırken motorun hangi fizik kanunları ile çalıştığını bilmek gerekmez

Bir bankaya kredi kartı için başvurulduğunda arka planda geçen brokrasiyi bilmek gerekmez.

“Türkler” kelimesi bir soyutlamadır.

Tek tek insanlar sayılmıyor ortak ve kültürel özellikler ile gruplandırılarak tek kelime ile milyonlarca insan ifade ediliyor.

Soyutlama – abstraction

Sınıf kullanımı açısından düşünülmeli

NTP açısından soyutlama

- Sınıfın iç yapısını bilmeye gerek yok
- Kullanılabilir metodları bilmek yeterli

Farklı seviyelerde soyutlama

- Sistem
- Komponent
- Sınıf

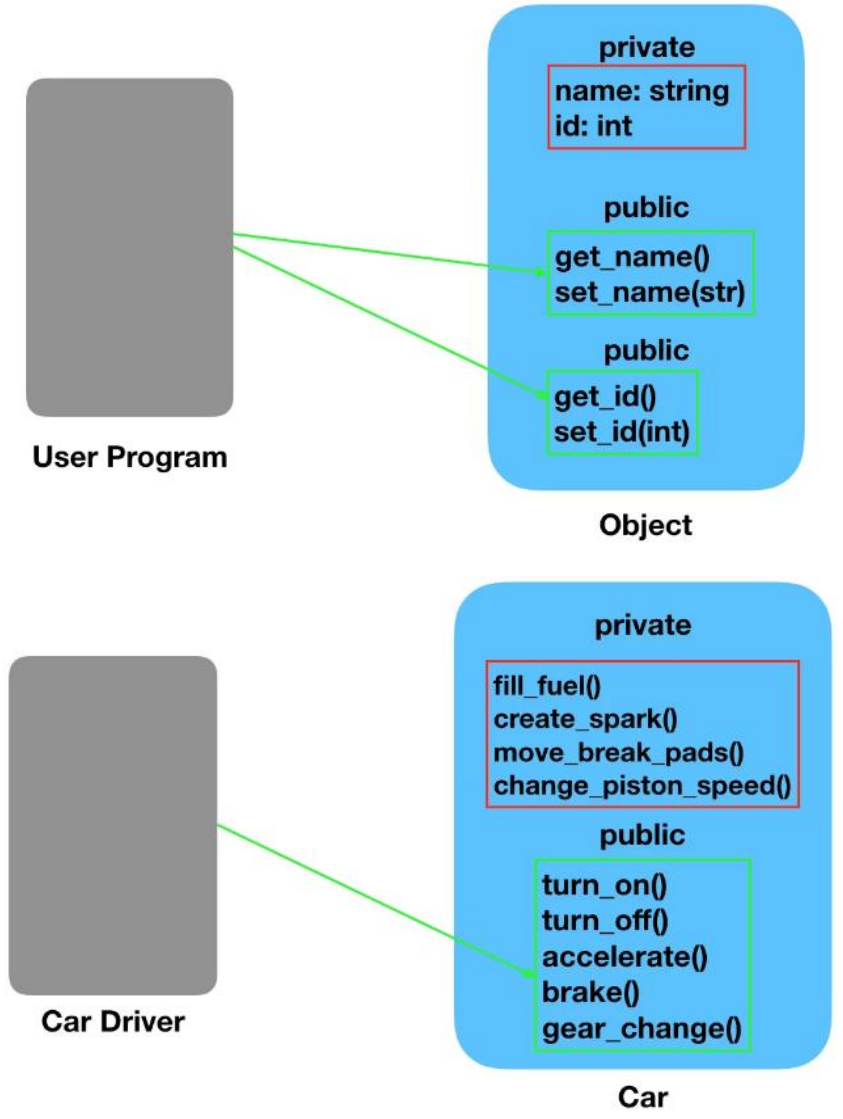
Kapsüllemeden farkı

- Kapsüllemenin amacı müdahaleyi düzenlemek
- Soyutlamanın amacı işlem detayını saklamak

Soyutlama çeşitleri

1. Veri soyutlama
   - Sınıf verileri ve işlenmesinin dış dünyaya kapalı olması
2. Süreç soyutlama
   - Süreçlerin nasıl gerçekleştiğine dair detaylar bilinmez
   - Sadece gerekli metotlar kullanılabilir

🧠 Sınıfı tasarlarken oluşturulan kapsülleme dış dünyada soyutlama oluşturur



Görsel Kaynağı: https://www.journaldev.com/33191/what-is-abstraction-in-oops

Java'da Soyutlama

- Interface
- Abstract sınıflar

```
public class CarTest {
	public static void main(String[] args) {
		Car car1 = new ManualCar();
		Car car2 = new AutomaticCar();

		car1.turnOnCar();
		car1.turnOffCar();
		System.out.println(car1.getCarType());

		car2.turnOnCar();
		car2.turnOffCar();
		System.out.println(car2.getCarType());

}
```

```
public interface Car {
	void turnOnCar();
	void turnOffCar();
	String getCarType();
}
```

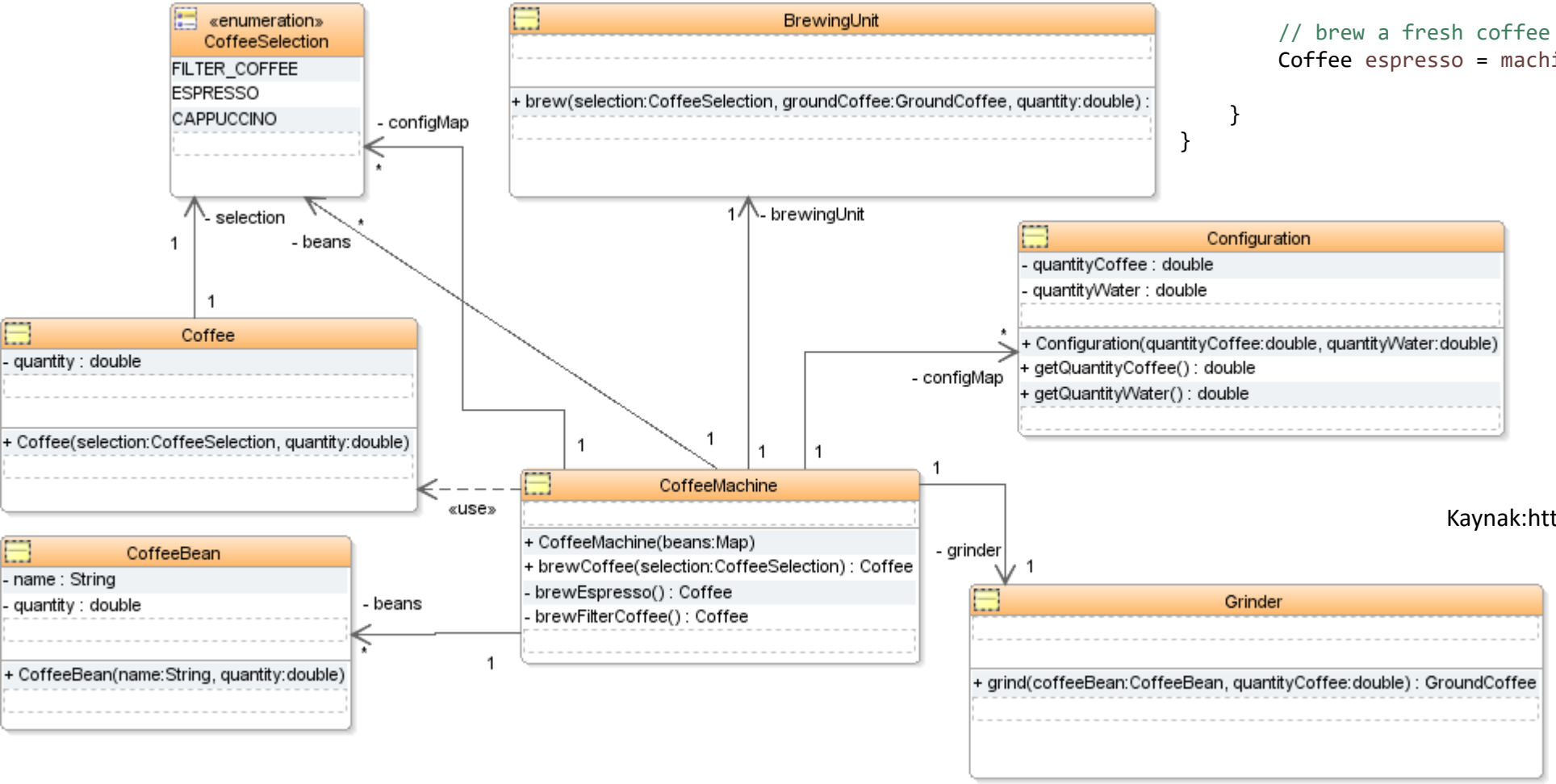
```
public class AutomaticCar implements Car {

	private String carType = "Automatic";

	public void turnOnCar() {
		System.out.println("turn on the automatic car");
	}

	public void turnOffCar() {
		System.out.println("turn off the automatic car");
	}

	public String getCarType() {
		return this.carType;
	}
}
```

```
public class ManualCar implements Car {

	private String carType = "Manual";

	public void turnOnCar() {
		System.out.println("turn on the manual car");
	}

	public void turnOffCar() {
		System.out.println("turn off the manual car");
	}

	public String getCarType() {
		return this.carType;
	}
}
```

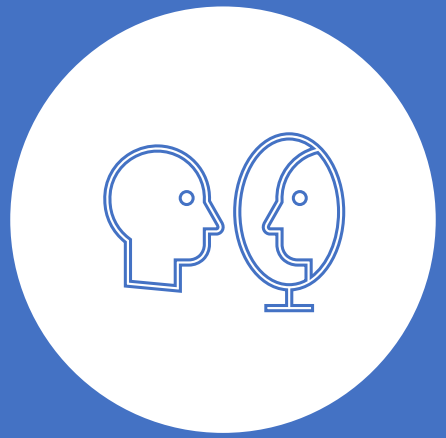
Soyutlama – abstraction

- Her sınıf kendi soyutlamasını gerçekleştirir
- Geliştiricisinin bilmesi yeterli

```
public class CoffeeApp {
    public static void main(String[] args) {
        // create a Map of available coffee beans
        Map<CoffeeSelection, CoffeeBean> beans
             = new HashMap<CoffeeSelection, CoffeeBean>();
        beans.put(CoffeeSelection.ESPRESSO,
            new CoffeeBean("My favorite espresso bean", 1000));
        beans.put(CoffeeSelection.FILTER_COFFEE,
            new CoffeeBean("My favorite filter coffee bean", 1000));

        // get a new CoffeeMachine object
        CoffeeMachine machine = new CoffeeMachine(beans);

        // brew a fresh coffee
        Coffee espresso = machine.brewCoffee(CoffeeSelection.ESPRESSO);

    }
}
```



Kaynak:https://stackify.com/oop-concept-abstraction/



Çok Biçimlilik – polymorphism

Aynı işlevin farklı nesnelerle yerine getirilmesi

Dr. Öğr. Üyesi Emre Bendeş

Çokbiçimlilik - polymorphism

```
Object x = null;
x = new Double(3.14);
x.toString();
x = new Date( );
x.toString();
```

```
Object x = new Something( );
System.out.println( x );
```

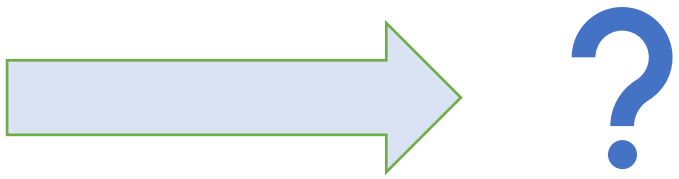
println(x), x.toString() metodunu çağırır.

```
Object a = .?.;
a.toString( );
a.run( );
```

Çalışır

Hata verir

Çok biçimlilik için esas olan nesne tipini bilmeden fonksiyonu çağırabilmektir.



Çokbiçimlilik - polymorphism

Çalışması için derleyicinin her zaman a nesnesinin run() metodunu sağlayacağını bilmesi gerekir.

```
a.run( );
```

Çağırdığımız farklı nesnelerin aynı metotlarının olduğunu garanti etmeliyiz.

Çokbiçimlilik - polymorphism

Java için metot varlığını garanti etmenin 2 yöntemi var:

1. Miras

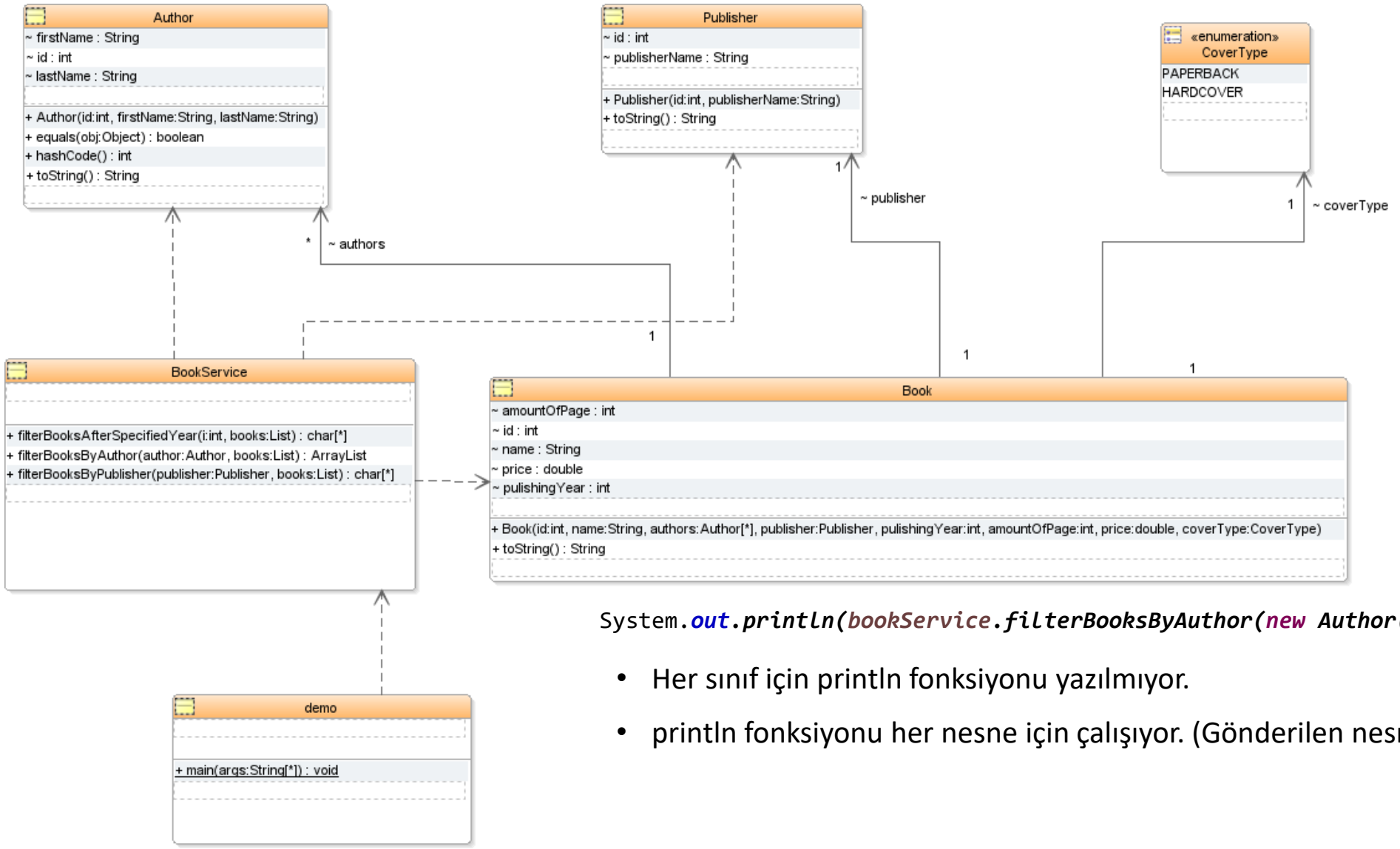
   Ana sınıfta ilgili metot bulundurulur.

   Alt sınıflarda bu metodun varlığı garanti olur

2. interface

   Yapısı gereği interface yi uygulayan sınıflar interface nin belirttiği metotları gerçekleştirmek zorundadır.

Bir örnek : **println** metodu




```
System.out.println(bookService.filterBooksByAuthor(new Author(1, "Jon", "Johnson"), books));
```

- Her sınıf için println fonksiyonu yazılmıyor.
- println fonksiyonu her nesne için çalışıyor. (Gönderilen nesnenin sınıfını bilmeden)

Çokbiçimlilik - polymorphism

- İleri uyumluluk (forward compatibility)
  - Geçmişte yazılan kodları gelecekte yazılacak kodları desteklemesi
  - Yeni kodlar eklenmesi gerektiğinde eski kodların değişmek zorunda kalmaması sağlar.
- Bir görevin yapılmasının farklı yolları olabilir
- Bu farklılıklar çevresel etkenler sebebiyle ya da problemin doğasından kaynaklanabilir
- Aynı sınıf ya da metodun farklı biçimlerde çalışabilmesini sağlar.
- Farklı seviyelerde çok biçimlilik oluşabilir
  - Miras
  - Soyutlama
  - Consructor overriding
  - Metod overriding-Statik bağlama (Static binding)
  - Aşırı yükleme (Overloading)-Dinamik bağlama (Dynamic binding)

Çokbiçimlilik - polimorphizm

- Binding (Bağlama) : method çağrısı ile metot gövdesinin ilişkilendirilmesini ifade eder

Static binding (early binding)

- Derleme zamanı
- **overload metot**ların bağlanması statiktir

Dynamic binding (late binding)

- Çalışma zamanı
- override metotların bağlanması dinamiktir
- Miras alınan sınıfla kullanılır.

```java
public class StaticBinding {
  public void staticbind() {
    System.out.println("This is static binding");
  }
  public void staticbind(String s) {
    System.out.println(s);
  }

  public static void main(String[] args) {
    StaticBinding sb = new StaticBinding();
    sb.staticbind();
    System.out.println("Static binding occurred.");
  }
}
```

Derleyici hangi metot un çağrıldığını derlerken belirleyebiliyor.

```java
class Dynamic {
     public void dynamicbind() {
         System.out.println("Original method dynamic bind of type
     Dynamic. ");
     }
}
public class DynamicBinding extends Dynamic {
     public void dynamicbind() {
          System.out.println("This is Dynamic Binding");
     }
     public static void main(String[] args) {
          DynamicBinding db = new DynamicBinding();
          db.dynamicbind();
     }
}
```

hangi metodun çağrıldığı çalışma zamanı belirleyebiliyor.

Çokbiçimlilik (polimorphizm) Çeşitleri

- Ad hoc polymorphism
- Parametric polymorphism
- Subtyping (inclusion)
- Operator

Ad hoc polymorphism

- Aynı metodu farklı uygulamalar ve farklı argümanlarla tanımlamak için kullanılan bir tekniktir.
  - Aşırı yükleme kullanılır (static binding)

```
public class AdHocPolymorphismExample {

    void sorting(int[] list) {
        Arrays.parallelSort(list);
        System.out.println("sıralamadan sonra tamsayılar: " + Arrays.toString(list) );
    }
    void sorting(String[] names) {
        Arrays.parallelSort(names);
        System.out.println("sıralamadan sonra isimler: " + Arrays.toString(names) );
    }

    public static void main(String[] args) {

        AdHocPolymorphismExample obj = new AdHocPolymorphismExample();
        int list[] = {2, 3, 1, 5, 4};
        obj.sorting(list);// Calling with integer array

        String[] names = {"ahmet", "veysel", "filiz", "gökhan"};
        obj.sorting(names);// Calling with String array
    }
}
```

Parametric polymorphism.

- Generik sınıflarla gerçekleştirilen çokbiçimlilik

```
Collection<Hayvan> hayvanlar = new ArrayList<Hayvan>();
Collection<Balik> baliklar = new ArrayList<Balik>();
```

Subtyping (inclusion)

- Dinamik bağlama kullanılır
- Override metotlar sayesinde farklı nesneler parametre olarak alınabilir.

```
abstract class Animal {
    abstract String talk();
}

class Cat extends Animal {
    String talk() {
        return "Miyaww!";
    }
}

class Dog extends Animal {
    String talk() {
        return "Havvv!";
    }
}

class Demo {
    static void letsHear(final Animal a) {
        println(a.talk());
    }

    static void main(String[] args) {
        letsHear(new Cat());
        letsHear(new Dog());
    }
}
```

Operator overloading

```
#include<iostream>
using namespace std;

class Complex {
private:
    int real, imag;
public:
    Complex(int r = 0, int i =0)  {real = r;   imag = i;}

    // This is automatically called when '+' is used with
    // between two Complex objects
    Complex operator + (Complex const &obj) {
         Complex res;
         res.real = real + obj.real;
         res.imag = imag + obj.imag;
         return res;
    }
    void print() { cout << real << " + i" << imag << endl; }
};

int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
}
```

Neden Çok Biçimlilik Kullanılmalı

- Modülariteyi artırır
  - Kod uyumu
- Kod esnekliği
- Standartta bağlı çeşitlilik