Behavioral - Iterator Design Pattern

foreach döngüsü, iterasyon mantığıyla çalışan bir mekanizmadır. Haliyle bu mekanizmanın kullandığı kaynak niteliğindeki yapıları genel olarak koleksiyonlar ve diziler olarak düşünebiliriz. Iterator design pattern için temel amaç elimizdeki veri yığını üzerinde ki bu array, list, stack vs olabilir, döngüsel işlemleri sağlayabilmektir

- Iterator
  Veri kümesi içerisinde dolaşmanın tüm şart ve imzasını bu arayüz belirlemektedir. Yani bir enumerator(sayıcı) görevi üstlenmektedir. Uzun lafın kısası, elimizdeki veri kümesi üzerinde döngü esnasında verileri/nesneleri elde edebilmemiz için gerekli işlemleri/kontrolleri/şartları/hususları tanımlar.

```
interface Iterator{
   public boolean hasNext();
   public Object next();
}
```

- Aggregate
  Veri kümesi içerisinde dolaşmak için bir Iterator interface'i tipinden Iterator yaratılmasını zorunlu tutan arayüzdür.

```
interface Container{
   Iterator getIterator();
}
```

- Class

```
class Person implements Container{
   public String[] names = {"Okyanus","Arzu"};
   @Override
   public Iterator getIterator() {
      return new PersonIterator();
   }
   private class PersonIterator implements Iterator{
      int index=0;
      @Override
      public boolean hasNext() {
         if (names.length>index){
            return true;
         }
         return false;
      }
      @Override
      public Object next() {
         if (this.hasNext()){
            return names[index++];
         }
         return false;
      }
   }
}
```

- Main

```
public static void main(String[] args) {
   Person person = new Person();
   Iterator iter = person.getIterator();
   while(iter.hasNext()){
      String name = (String) iter.next();
      System.out.println(name);
   }
}
```

PersonIterator inner class'ı içerisinde kendimize has algoritmalar oluşturup kullanabiliriz