Lambda ifadeleri

Gerekli olanlar :

1. Tek metotlu interface (fonksiyonel interface)
2. Ananoim sınıftaki gibi "= new" ifadesi yerine " ()->" ifadesi kullanarak aslında anonim bir sınıf oluşturulmakta.

```
public class Calculator {

    interface IntegerMath {
        int operation(int a, int b);
    }

    public int operateBinary(int a, int b, IntegerMath op) {
        return op.operation(a, b);
    }

    public static void main(String... args) {

        Calculator myApp = new Calculator();
        IntegerMath addition = (a, b) -> a + b;
        IntegerMath subtraction = (a, b) -> a - b;
        System.out.println("40 + 2 = " + myApp.operateBinary(40, 2, addition));
        System.out.println("20 - 10 = " + myApp.operateBinary(20, 10, subtraction));
    }
}
```

Kaynak: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

Bir yöneticinin, sosyal ağ uygulamasının belirli kriterleri karşılayan üyelerine mesaj göndermek gibi her türlü eylemi gerçekleştirmesine olanak tanıyan bir özellik oluşturmak istiyorsunuz.

Alan	**Tanım**
İsim	Seçilen üyeler üzerinde işlem gerçekleştirin
Birincil aktör	yönetici
ön koşullar	Yönetici sistemde oturum açmıştır.
Sonlanma koşulları	Eylem, yalnızca belirtilen ölçütlere uyan üyeler üzerinde gerçekleştirilir.
Ana Başarı Senaryosu	1.Yönetici, belirli bir eylemin gerçekleştirileceği üyelerin ölçütlerini belirtir. 2.Yönetici, seçilen üyeler üzerinde gerçekleştirilecek bir eylemi belirtir. 3.Yönetici **Gönder** düğmesini seçer. 4.Sistem, belirtilen kriterlere uyan tüm üyeleri bulur. 5.Sistem, eşleşen tüm üyelerde belirtilen eylemi gerçekleştirir.
Uzantılar	Yönetici, gerçekleştirilecek eylemi belirtmeden veya Gönder düğmesini seçmeden önce belirtilen ölçütlerle eşleşen üyeleri ön izleme seçeneğine sahiptir
Oluşma sıklığı	Gün içinde birçok kez.

```
public class Person {

    public enum Sex {
        MALE, FEMALE
    }

    String name;
    LocalDate birthday;
    Sex gender;
    String emailAddress;

    Person(String nameArg, LocalDate birthdayArg,
        Sex genderArg, String emailArg) {
        name = nameArg;
        birthday = birthdayArg;
        gender = genderArg;
        emailAddress = emailArg;
    }

    public int getAge() {
        return birthday
            .until(IsoChronology.INSTANCE.dateNow())
            .getYears();
    }

    public void printPerson() {
      System.out.println(name + ", " + this.getAge());
    }

    public Sex getGender() {
        return gender;
    }

    public String getName() {
        return name;
    }

    public String getEmailAddress() {
        return emailAddress;
    }

    public LocalDate getBirthday() {
        return birthday;
    }

    public static int compareByAge(Person a, Person b) {
        return a.birthday.compareTo(b.birthday);
    }
}
```

```
public static List<Person> createRoster() {

    List<Person> roster = new ArrayList<>();
    roster.add(
        new Person(
        "Fred",
        IsoChronology.INSTANCE.date(1980, 6, 20),
        Person.Sex.MALE,
        "fred@example.com"));
    roster.add(
        new Person(
        "Jane",
        IsoChronology.INSTANCE.date(1990, 7, 15),
        Person.Sex.FEMALE, "jane@example.com"));
    roster.add(
        new Person(
        "George",
        IsoChronology.INSTANCE.date(1991, 8, 13),
        Person.Sex.MALE, "george@example.com"));
    roster.add(
        new Person(
        "Bob",
        IsoChronology.INSTANCE.date(2000, 9, 12),
        Person.Sex.MALE, "bob@example.com"));

    return roster;
}
```

Basit bir yaklaşım, birkaç yöntem oluşturmaktır;

her yöntem, cinsiyet veya yaş gibi bir özellikle eşleşen üyeleri arar.

```
public class RosterTest {
    public static void printPersonsOlderThan(List<Person> roster, int age) {
        for (Person p : roster) {
            if (p.getAge() >= age) {
                p.printPerson();
            }
        }
    }
    public static void printPersonsWithinAgeRange(
        List<Person> roster, int low, int high) {
        for (Person p : roster) {
            if (low <= p.getAge() && p.getAge() < high) {
                p.printPerson();
            }
        }
    }
    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }
        System.out.println("Persons older than 20:");
        printPersonsOlderThan(roster, 20);
        System.out.println("Persons between the ages of 14 and 30:");
        printPersonsWithinAgeRange(roster, 14, 30);
    }
}
```

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons older than 20:
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons between the ages of 14 and 30:
Bob, 22
```

Bu yaklaşım, uygulamanızın *kırılgan* hale gelmesine neden olabilir;

- güncellemelerin (daha yeni veri türleri gibi) kullanıma sunulması nedeniyle bir uygulamanın çalışmama olasılığı.
- Person sınıfında yaş yerine Doğum yılı tutup yaşı hesaplama yaklaşımına geçildiğini düşünelim
- Ayrıca, bu yaklaşım gereksiz yere kısıtlayıcıdır; Örneğin, belirli bir yaştan daha genç üyeleri yazdırmak isteseydiniz ne olurdu?
- Belirli bir cinsiyetteki üyeleri veya belirli bir cinsiyet ve yaş aralığının bir kombinasyonunu yazdırmak isterseniz ne olur?
- Person Sınıfı değiştirmeye ve ilişki durumu veya coğrafi konum gibi başka özellikler eklemeye karar verirseniz ne olur ?
- Bu yöntemden daha genel olmasına rağmen printPersonOlderThen olası her arama sorgusu için ayrı bir yöntem oluşturmaya çalışmak yine de kırılgan koda yol açabilir

Bunun yerine, farklı bir sınıfta aramak istediğiniz ölçütleri belirten kodu ayırabilirsiniz.

3. Yaklaşım

```
public class Demo {

    interface CheckPerson {
        boolean test(Person p);
    }

    public static void printPersons(List<Person> roster, CheckPerson tester) {
        for (Person p : roster) {
            if (tester.test(p)) {
                p.printPerson();
            }
        }
    }

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }

        System.out.println("Persons who are eligible for Selective Service:");
        class CheckPersonEligibleForSelectiveService implements CheckPerson {
           public boolean test(Person p) {
                return p.getGender() == Person.Sex.MALE
                    && p.getAge() >= 18
                    && p.getAge() <= 25;
            }
        }

        printPersons(roster, new CheckPersonEligibleForSelectiveService());
    }
}
```

- **`printPersons`** metodu içerisinde bir şart yok
  - Kendini şartın detayından soyutlamış
  - Şartlar dışarıdan gönderiliyor
- main içerisinde şart **`CheckPerson`** i uygulayan bir yerel sınıf nesnesi ile **`printPersons`** metoduna gönderilmiş
- Kod daha az kırılgan
- Ancak Person sınıfı içindeki tanımlama değiştiğinde ilave kod yazmaya gerek duyar
  - Yeni bir interface
  - Her arama için yeni bir yerel sınıf

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service:
Bob, 22
```

4. Yaklaşım

- 3 yaklaşımda yeni yerel sınıfı oluşturma işlemini ortadan kaldırmak için anonim sınıf kullanılabilir.
- Gerekli kod miktarını azaltır. Kod sadeliği sağlar (göreceli)
- Filtreleme bir parametre içerisinde gerçekleşmekte
- Her arama için bir sınıf yazmaya gerek kalmadı

```
public class Demo {

    interface CheckPerson {
        boolean test(Person p);
    }

    public static void printPersons(List<Person> roster, CheckPerson tester) {
        for (Person p : roster) {
            if (tester.test(p)) {
                p.printPerson();
            }
        }
    }

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }

        System.out.println("Persons who are eligible for Selective Service " + "(anonymous class):");
        printPersons(
            roster,
            new CheckPerson() {
                public boolean test(Person p) {
                    return p.getGender() == Person.Sex.MALE
                        && p.getAge() >= 18
                        && p.getAge() <= 25;
                }
            }
        );
    }
}
```

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service (anonymous class):
Bob, 22
```

5. Yaklaşım

- İnterface bir metot içerdiği (fonksiyonel interface) için Lambda ifadesi kullanılabilir.
  - Kod sadeliği artırılmış olur.
- **`printPersons`** Metodunda herhangi bir değişiklik yok
- Lambda ifadesi sanki nesne değil de metot gönderiyormuş gibi kodu sadeleştiriyor.
  - Aslında tek metotlu bir interface tipinde anonim bir sınıf oluşturuluyor.

```
public class Demo {

    interface CheckPerson {
        boolean test(Person p);
    }

    public static void printPersons(List<Person> roster, CheckPerson tester) {
        for (Person p : roster) {
            if (tester.test(p)) {
                p.printPerson();
            }
        }
    }

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }

        System.out.println("Persons who are eligible for Selective Service " + "lambda expression):");

        printPersons(
            roster,
            (Person p) -> p.getGender() == Person.Sex.MALE
                && p.getAge() >= 18
                && p.getAge() <= 25
        );
    }
}
```

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service (lambda expression):
Bob, 22
```

6. Yaklaşım

```
import java.util.function.Predicate;

public class Demo {


    public static void printPersons(
        List<Person> roster, Predicate<Person> tester) {
        for (Person p : roster) {
            if (tester.test(p)) {
                p.printPerson();
            }
        }
    }

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }

        System.out.println("Persons who are eligible for Selective Service " + "(with Predicate parameter):");

        printPersons(
            roster,
            p -> p.getGender() == Person.Sex.MALE
                && p.getAge() >= 18
                && p.getAge() <= 25
        );
    }
}
```

```
interface CheckPerson {

        boolean test(Person p);

}
```

- Bu çok standart bir interface.
  - Sadece bir parametre alan, boolean dönderen tek metotlu interface

- JDK java.util.function paketi içinde birçok standart fonksiyonel interface tanımı sunmaktadır.
  - Tekrar yazmak gerekmeden import edip kullanabiliriz.
- **`Predicate`** isimli boolean tipi döndüren bir metotlu generic bir interface java.util.function içerisinde tanımlı :

```
interface Predicate<T> {

    boolean test(T t);

}
```

- **`printPersons`** metodunun parametresi değiştiğine dikkat edin

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service (with Predicate parameter):
Bob, 22
```

7.1. Yaklaşım

- Şartlar sağlandığında ekrana basılıyordu.
- Şart sağlandığında ne yapılacağı da belirlenebilir mi ?
- Bu işlem başka bir lambda ifadesi ile gerçekleştirilebilir.
- **processPersons** metodu kendini tamamen soyutlamış olur
- Bu amaçla **Consumer** interfacesi kullanılabilir.

```
public interface Consumer<T> {

    void accept(T t);

}
```

```
public class Demo {


    public static void processPersons(
        List<Person> roster,
        Predicate<Person> tester,
        Consumer<Person> block)
    {
        for (Person p : roster) {
            if (tester.test(p)) {
                block.accept(p);
            }
        }
    }

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }

        System.out.println("Persons who are eligible for Selective Service " + "(with Predicate and Consumer parameters):");

        processPersons(
            roster,
            p -> p.getGender() == Person.Sex.MALE
                && p.getAge() >= 18
                && p.getAge() <= 25,
            p -> p.printPerson()
        );
    }
}
```

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service (with Predicate and Consumer parameters):
Bob, 22
```

7.2. Yaklaşım

- Ya üyelerinizin profillerini yazdırmaktan daha fazlasını yapmak istiyorsanız?
- Diyelim ki üyelerin profillerini doğrulamak veya iletişim bilgilerini almak istiyorsunuz?
- Bu durumda, bir değer döndüren soyut bir yöntem içeren bir fonksiyonel interface ihtiyacınız vardır.

```
public interface Function<T, R> {

    R apply(T t);

}
```

- **`processPersonsWithFunction`** yöntemi
  - mapper parametresi tarafından belirtilen verileri alır
  - ardından block parametresi tarafından belirtilen veriler üzerinde bir eylem gerçekleştirir

```
public class Demo {


    public static void processPersonsWithFunction(
        List<Person> roster,
        Predicate<Person> tester,
        Function<Person, String> mapper,
        Consumer<String> block)
    {
        for (Person p : roster) {
            if (tester.test(p)) {
                String data = mapper.apply(p);
                block.accept(data);
            }
        }
    }

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }
        System.out.println("Persons who are eligible for Selective Service " + "(with Predicate, Function, and Consumer parameters):");
        processPersonsWithFunction(
            roster,
            p -> p.getGender() == Person.Sex.MALE
                && p.getAge() >= 18
                && p.getAge() <= 25,
            p -> p.getEmailAddress(),
            email -> System.out.println(email)
        );
    }
}
```

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service (with Predicate, Function, and Consumer parameters):
bob@example.com
```

8. Yaklaşım

```
public class Demo {


    public static <X, Y> void processElements(
        Iterable<X> source,
        Predicate<X> tester,
        Function<X, Y> mapper,
        Consumer<Y> block)
    {
            for (X p : source) {
                if (tester.test(p)) {
                    Y data = mapper.apply(p);
                    block.accept(data);
                }
            }
    }

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }

        System.out.println("Persons who are eligible for Selective Service " + "(generic version):");
        processElements(
            roster,
            p -> p.getGender() == Person.Sex.MALE
                && p.getAge() >= 18
                && p.getAge() <= 25,
            p -> p.getEmailAddress(),
            email -> System.out.println(email)
        );

     }
}
```

- Generik ler ile metot daha fazla soyutlaştırılabilir.
  - Metodun içinde hangi işlem yapılacağı kime yapılacağı filtrelemenin nasıl yapılacağı belli değil

Şu dört işlem gerçekleşmekte:

1. source parametresinden nesnelerin kaynağını tespit edilir.

   Bu örnekte Person sınıfı roster isimli List generik nesnesinden tespit edildi

   List iterable olduğu için döngüde kullanılabildi (**for (X p : source)**)
2. Predicate nesnesi tester ile nesneler filtrelendi

   Bu örnekte Predicate lambda ifadesi ile kullanıcılar üzerine filtre uyguluyor.
3. Bir önceki adımda filtrelenen her nesne Function nesnesi mapper ile belirlenen bir değere dönüştürüldü.

   Bu örnekte People nesnesi alınıp string mail adresi elde ediliyor.
4. Consumer nesnesi block ile her değer üzerine bir işlem uygulandı

   Bu örnekte alınan değer ekrana basılıyor.

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service (generic version):
bob@example.com
```

9. Yaklaşım

```
public class Demo {

    public static void main(String[] args) {
        List<Person> roster = Person.createRoster();
        for (Person p : roster) {
            p.printPerson();
        }

        System.out.println("Persons who are eligible for Selective Service " +
                "(with bulk data operations):");

            roster
                .stream()
                .filter(
                    p -> p.getGender() == Person.Sex.MALE
                        && p.getAge() >= 18
                        && p.getAge() <= 25)
                .map(p -> p.getEmailAddress())
                .forEach(email -> System.out.println(email));

    }
}
```

```
Fred, 42
Jane, 32
George, 31
Bob, 22
Persons who are eligible for Selective Service (with bulk data operations):
bob@example.com
```