Creational Pattern - Builder Design Pattern (2. Örneği mutlaka incele)

Çok basit bir örnek ile açıklayalım.Person isimli bir class'ımız var ve bu sınıfın içinde field'larımız var

```
class Person{
   private String firstName;
   private String lastName;
   private String address;
}
```

Bizim bu class'ın field'lerinden sadece firstName ve lastName'i kullanacağımızı düşünelim. Bunun için bir constructor inşa ettik ve şimdilik problemi çözdüm. Başka bir senaryo da sadece firstName'e ihtiyacımız oldu, tekrardan yeni bir constructor oluşturduk. Bu senaryolar böyle uzayıp gittiğinde işin içinden çıkılmaz bir hal oluşmaya başlar. Bunun önüne geçmek içinde Builder Design Pattern kullanılır

```
interface HousePlan {
   void setBasement(String basement); //temel
   void setStructure(String structure); //yapı,bina
   void setRoof(String roof); //çatı
   void setInterior(String interior); //iç mekan
}
```

Setter'ları görüldüğü gibi HousePlan içerisinde tanımlıyoruz.House class'ı HousePlan'ı implemente ediyor

```
class House implements HousePlan {
   private String basement;
   private String structure;
   private String roof;
   private String interior;
   @Override
   public void setBasement(String basement) {
      this.basement = basement;
   }
   @Override
   public void setStructure(String structure) {
      this.structure = structure;
   }
   @Override
   public void setRoof(String roof) {
      this.roof = roof;
   }
   @Override
   public void setInterior(String interior) {
      this.interior = interior;
   }

   @Override
   public String toString() {
      return "House{" +
            "basement='" + basement + '\'' +
            ", structure='" + structure + '\'' +
            ", roof='" + roof + '\'' +
            ", interior='" + interior + '\'' +
            '}';
   }
}
```

```
//Builder
interface HouseBuilder{
    void buildBasement();
    void buildStructure();
    void buildRoof();
    void buildInterior();
    House getHouse();
}
```

Her bir HouseBuilder nesnesi HouseBuilder nesnesini implemente edecek

```
class IglooHouseBuilder implements HouseBuilder{
    private House house;
    public IglooHouseBuilder(){
        this.house = new House();
    }
    @Override
    public void buildBasement() {
        house.setBasement("Ice Bars");
    }
    @Override
    public void buildStructure() {
        house.setStructure("Ice Blocks");
    }
    @Override
    public void buildRoof() {
        house.setRoof("Ice Dome");
    }
    @Override
    public void buildInterior() {
        house.setInterior("Ice carvings");
    }
    @Override
    public House getHouse() {
        return this.house;
    }
}
```

```
class TipiHouseBuilder implements HouseBuilder{
    private House house;
    public TipiHouseBuilder(){
        this.house = new House();
    }
    @Override
    public void buildBasement() {
        house.setBasement("Wooden Poles");
    }
    @Override
    public void buildStructure() {
        house.setStructure("Wood and ice");
    }
    @Override
    public void buildRoof() {
        house.setRoof("Wood,caribou and seal skins");
    }
    @Override
    public void buildInterior() {
        house.setInterior("Fire wood");
    }
    @Override
    public House getHouse() {
        return house;
    }
}
```

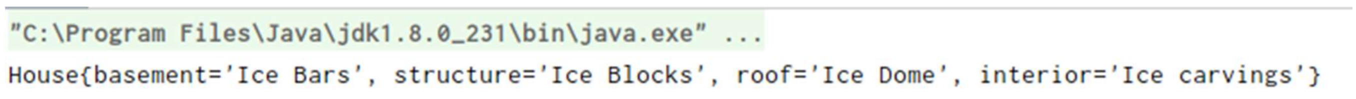
Şimdi bu işleri yönetecek olan director interface’i

```java
interface Director{
    House getHouse();
    void ConstructHouse();
}

//Concrete Director
class CivilEngineer implements Director{
    private HouseBuilder houseBuilder;
    public CivilEngineer(HouseBuilder houseBuilder){
        this.houseBuilder = houseBuilder;
    }
    @Override
    public House getHouse() {
        return this.houseBuilder.getHouse();
    }
    @Override
    public void ConstructHouse() {
        this.houseBuilder.buildBasement();
        this.houseBuilder.buildStructure();
        this.houseBuilder.buildInterior();
        this.houseBuilder.buildRoof();
    }
}
```

Main Class;

```java
class Test{
    public static void main(String[] args) {
        HouseBuilder iglooHouseBuilder = new IglooHouseBuilder();
        Director director = new CivilEngineer(iglooHouseBuilder);
        director.ConstructHouse();
        House house = director.getHouse();
        System.out.println(house);
    }
}
```



```
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...
House{basement='Ice Bars', structure='Ice Blocks', roof='Ice Dome', interior='Ice carvings'}
```

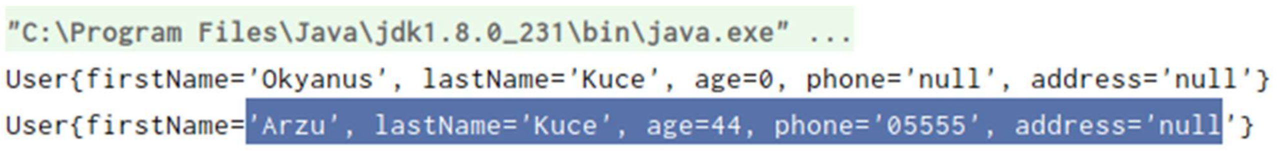
Creational Pattern - Builder Design Pattern (Static class ile) - User class'ı içerisinde static bir class olarak builder tanımlıyoruz ve istediğimiz gibi nesneyi tanımlayabiliyoruz.

```java
class User {
   private final String firstName; //required
   private final String lastName; //required
   private final int age; //optional
   private final String phone; //optional
   private final String address; //optional
   //private constructor
   private User(UserBuilder userBuilder){
      this.firstName = userBuilder.firstName;
      this.lastName = userBuilder.lastName;
      this.age = userBuilder.age;
      this.phone = userBuilder.phone;
      this.address = userBuilder.address;
   }
   //All getter, and NO setter to provde immutability
   public String getFirstName() {
      return firstName;
   }
   public String getLastName() {
      return lastName;
   }
   public int getAge() {
      return age;
   }
   public String getPhone() {
      return phone;
   }
   public String getAddress() {
      return address;
   }
   @Override
   public String toString() {
      return "User{" +
            "firstName='" + firstName + '\'' +
            ", lastName='" + lastName + '\'' +
            ", age=" + age +
            ", phone='" + phone + '\'' +
            ", address='" + address + '\'' +
            '}';
   }
   public static class UserBuilder{
      private final String firstName;//required
      private final String lastName;//required
      private int age;//optional
      private String phone;//optional
      private String address;//optional
      //Required alanlar
      public UserBuilder(String firstName, String lastName) {
         this.firstName = firstName;
         this.lastName = lastName;
      }
      //Optional
      public UserBuilder age(int age) {
         this.age = age;
         return this;
      }
      //Optional
      public UserBuilder phone(String phone) {
         this.phone = phone;
         return this;
      }
      //Optional
```

```
            public UserBuilder address(String address) {
                this.address = address;
                return this;
            }
            public User build(){
                User user = new User(this);
                validateUserObject(user);
                return user;
            }
            private void validateUserObject(User user) {
                //Do some basic validations to check
                //if user object does not break any assumption of system
            }
        }
    }
```

Main Class;

```
class Test{
    public static void main(String[] args) {
        User okyanus = new User.UserBuilder("Okyanus","Kuce").build();
        User arzu = new User.UserBuilder("Arzu","Kuce").age(44).phone("05555").build();
        System.out.println(okyanus);
        System.out.println(arzu);
    }
}
```



```
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...
User{firstName='Okyanus', lastName='Kuce', age=0, phone='null', address='null'}
User{firstName='Arzu', lastName='Kuce', age=44, phone='05555', address='null'}
```