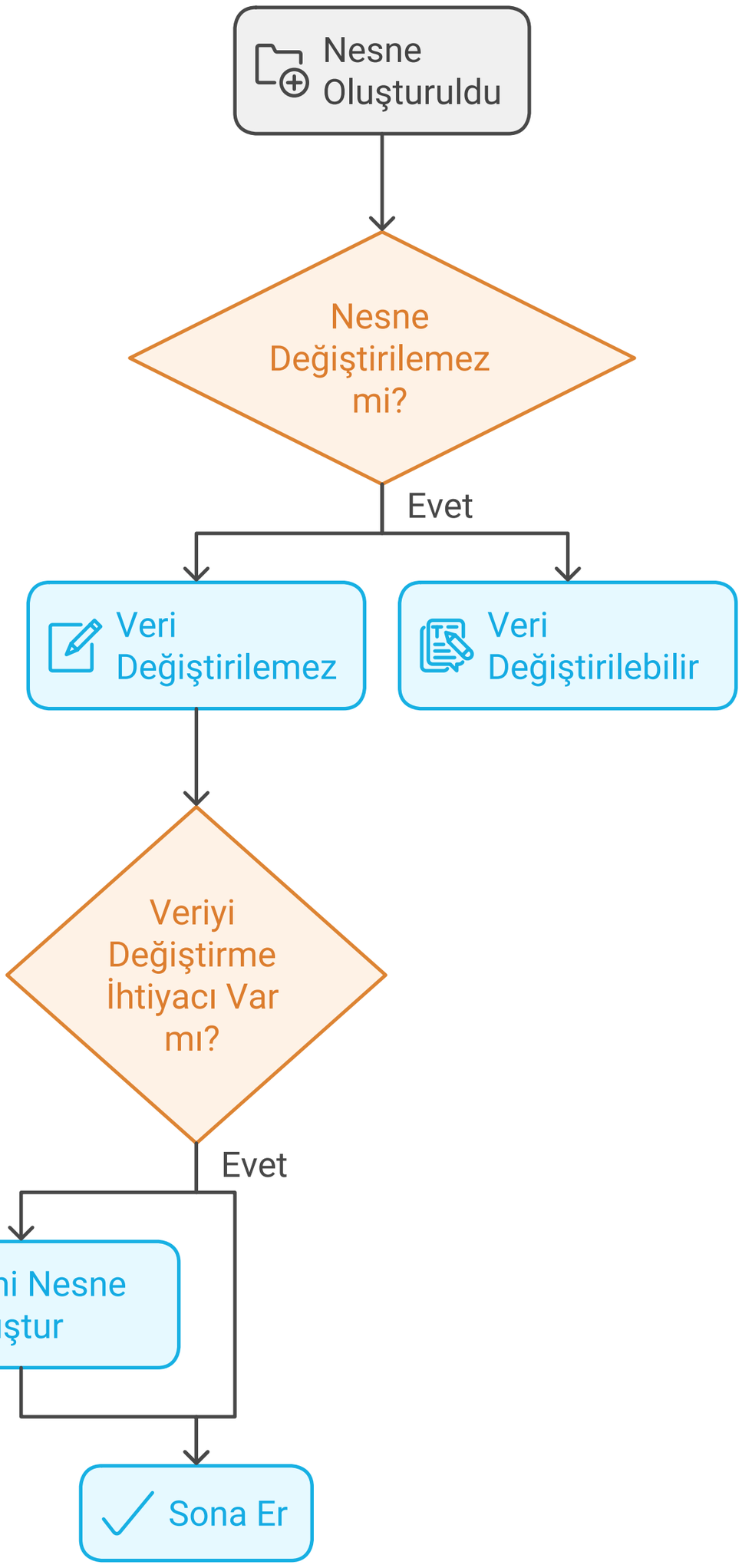
Java'da Immutable (Değişmez) Nesneler

Java'da immutable (değişmez) nesneler, bir nesne oluşturulduktan sonra onun durumunun (state) değiştirilemediği nesnelerdir. Bu özellik, yazılım geliştirmede birçok avantaj sağlar, özellikle çok iş parçacıklı (multithreaded) ortamlarda. Bu belgede, Java'da immutable nesnelerin nasıl oluşturulacağını ve kullanılacağını adım adım inceleyeceğiz.

1. Immutable Nedir?

Immutable, bir nesnenin oluşturulduktan sonra değiştirilemez olduğu anlamına gelir. Yani, bir nesne oluşturulduğunda, onun içindeki veriler değiştirilemez. Eğer nesne üzerinde bir değişiklik yapmak isterseniz, yeni bir nesne oluşturmanız gerekir.



2. Immutable Nesne Oluşturma Adımları

2.1. Sınıfı Tanımlama

Öncelikle, immutable bir sınıf tanımlamalıyız. Bu sınıfın tüm alanları **final** olarak tanımlanmalıdır. Bu, alanların yalnızca bir kez atanabileceği anlamına gelir.

```
public final class ImmutableClass {
    private final String name;
    private final int age;

    public ImmutableClass(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

2.2. Getter Metotları

Immutable sınıflarda, alanlara erişim sağlamak için yalnızca getter metotları tanımlanmalıdır. Setter metotları kullanılmamalıdır, çünkü bu metotlar nesnenin durumunu değiştirebilir.

```
public String getName() {
    return name;
}

public int getAge() {
    return age;
}
```

2.3. Derin Kopyalama (Deep Copy)

Eğer sınıfınızda mutable (değişebilir) nesneler varsa, bu nesnelerin derin kopyalarını almanız gerekir. Bu, dışarıdan nesnenin durumunu değiştirilmesini engeller.

```
private final List<String> hobbies;

public ImmutableClass(String name, int age, List<String> hobbies) {
    this.name = name;
    this.age = age;
    this.hobbies = new ArrayList<>(hobbies); // Derin kopyalama
}

public List<String> getHobbies() {
    return new ArrayList<>(hobbies); // Derin kopyalama
}
```

3. Immutable Sınıf Kullanımı

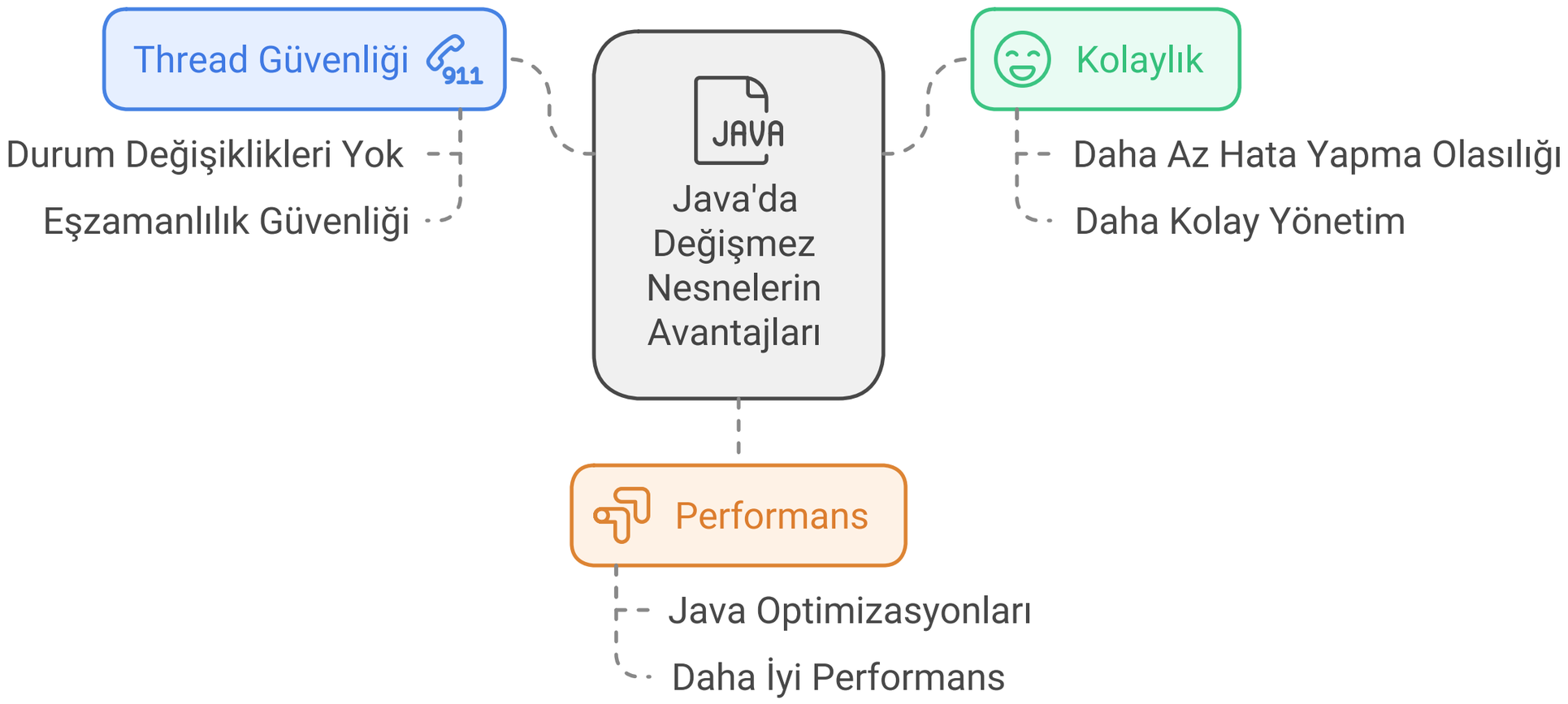
Immutable sınıfı kullanmak oldukça basittir. Aşağıda bir örnek verilmiştir:

```
public class Main {
    public static void main(String[] args) {
        List<String> hobbies = Arrays.asList("Reading", "Traveling");
        ImmutableClass person = new ImmutableClass("Alice", 30, hobbies);

        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
        System.out.println("Hobbies: " + person.getHobbies());
    }
}
```

4. Avantajları

- **Thread Güvenliği**: Immutable nesneler, çok iş parçacıklı ortamlarda güvenlidir çünkü durumları değişmez.
- **Kolaylık**: Immutable nesneler, daha az hata yapma olasılığı ile daha kolay yönetilebilir.
- **Performans**: Java'nın optimizasyonları sayesinde, immutable nesneler daha iyi performans gösterebilir.



5. Sonuç

Java'da immutable nesneler, yazılım geliştirme sürecinde önemli bir rol oynamaktadır. Bu belgede, immutable nesnelerin nasıl oluşturulacağı ve kullanılacağı adım adım açıklanmıştır. Immutable nesnelerin sağladığı avantajlar, yazılım projelerinde daha güvenli ve sürdürülebilir bir yapı oluşturmanıza yardımcı olabilir.