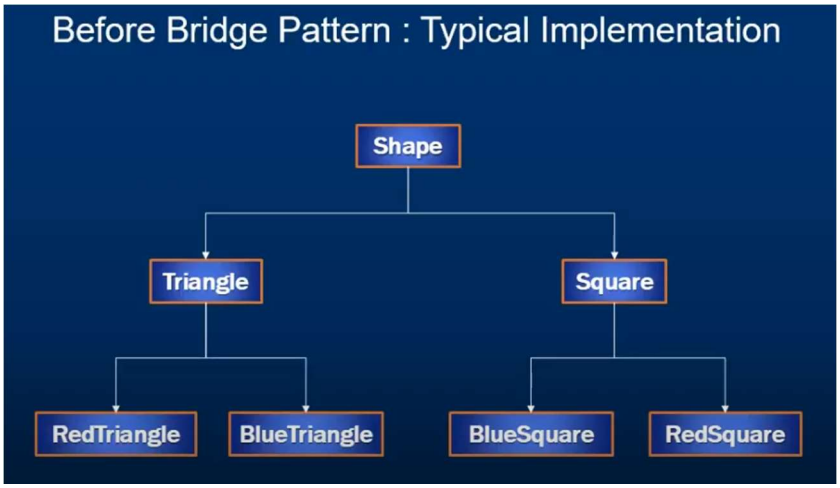
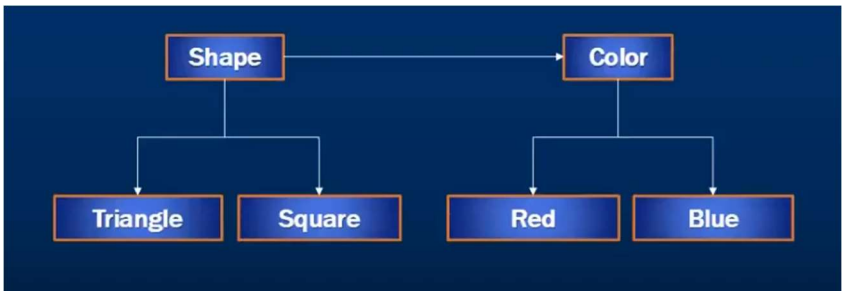
Structural - Bridge Design Pattern



Bridge Pattern kullanılmadan önce Shape base class'ından Triange ve Square adlı iki concrete class türüyor, bunlarda kendi aralarında BlueSquare ve RedSquare örneğinde olduğu gibi ayrışıyorlar. Bridge Design Pattern diyor ki; soyutlaşmış (abstract) bir yapıyı, implementasyondan ayır. Böylece bağımsız olarak geliştirilebilir iki yapı elde edersin.

Bridge pattern kullanıldığında;



Şöyle bir örnek üzerinden devam edelim;

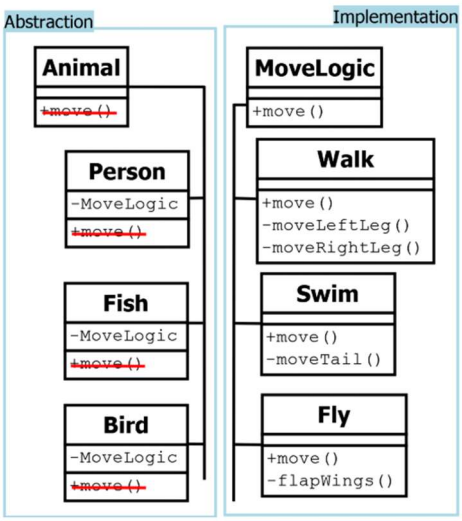
(Abstraction)

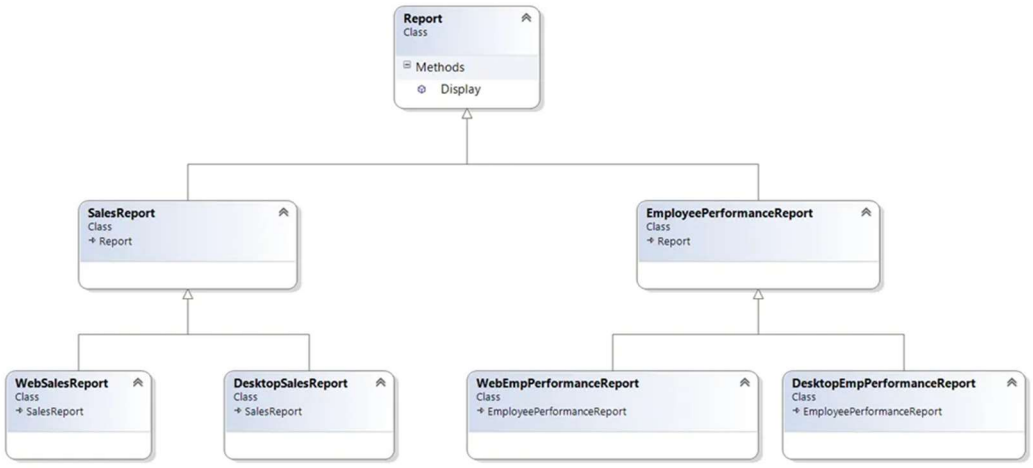
- İnsan yürüyerek hareket eder
- Bir balık yüzerek hareket eder
- Bir kuş uçarak hareket eder

Farklı yollarla hareket eden 3 nesne. Burada bridge pattern'in amacı move mantığını abstraction'dan ayırmaktır.Her bir nesne için move methodu başka şekilde implemente edilmektedir

(Implementation)

Move mantığı abstraction'dan ayrıştırılır





Bu örnek üzerinden devam edecek olursak. Bridge Design Pattern diyor ki; soyutlaşmış (abstract) bir yapıyı, implementasyondan ayır. Böylece bağımsız olarak geliştirilebilir iki yapı elde edersin.Implementasyon'dan kastettiği şey en son da elde edeceğimiz sınıftır yani bu örnekte DesktopSalesReport. WebEmpPerformanceReport ile DesktopSalesReport sınıfları ortak bir atadan türediklerine göre birbirleriyle akrabalar öyle değil mi? Belirttiğim iki sınıf geliştirilebilir bir modelde olabilmesi için akraba **olmamalılar.** Nitekim Web formatında oluşturulmuş çalışan performans raporu başka bir şey, masaüstü formatında oluşturulmuş satış raporu ise bambaşka. Bu iki sınıf da **farklı iki formatta** bir rapordur. O halde bakın ne açığa çıktı! **Rapor Formatı, bu modelde soyutlaşmış bir yapıdır.** Yani, satış ya da çalışan performans raporunu oluştururken, hangi formatta (Masaüstü veya web) kaydetmesi gerektiğini söylemem yeterli olacak. Bu durumu ayarlamak için **ReportFormat** isminde bir interface oluşturuyorum ve ilgili formatları bu interface'den implemente ediyorum

```
interface ReportFormat{
   void generate();
}
```

Interface'i implemente eden Concrete class'lar

```
class WebFormat implements ReportFormat{
   public void generate(){
      System.out.println("Generate to the Web format");
   }
}

class DesktopFormat implements ReportFormat{
   public void generate(){
      System.out.println("Generate to the Desktop format");
   }
}
```

Abstraction :

```
abstract class Report{
    protected ReportFormat reportFormat;
    public Report(ReportFormat newReportFormat){
        reportFormat = newReportFormat;
    }

    void display(){
        reportFormat.generate();
    }
}
```

Abstraction'ı extends eden class'lar:

```
abstract class Report{
    protected ReportFormat reportFormat;
    public Report(ReportFormat newReportFormat){
        reportFormat = newReportFormat;
    }

    void display(){
        reportFormat.generate();
    }
}

class SalesReport extends Report{
    public SalesReport(ReportFormat newReportFormat) {
        super(newReportFormat);
    }

    @Override
    void display() {
        System.out.println("---Sales report---");
        super.display();
    }
}

class EmployeePerformaceReport extends Report{
    public EmployeePerformaceReport(ReportFormat newReportFormat) {
        super(newReportFormat);
    }

    @Override
    void display() {
        System.out.println("---Employee performance report---");
        super.display();
    }
}
```

Main:

```
public static void main(String[] args) {
    Report report = new SalesReport(new DesktopFormat());
    report.display();
    Report reportTwo = new EmployeePerformaceReport(new DesktopFormat());
    reportTwo.display();
}
```