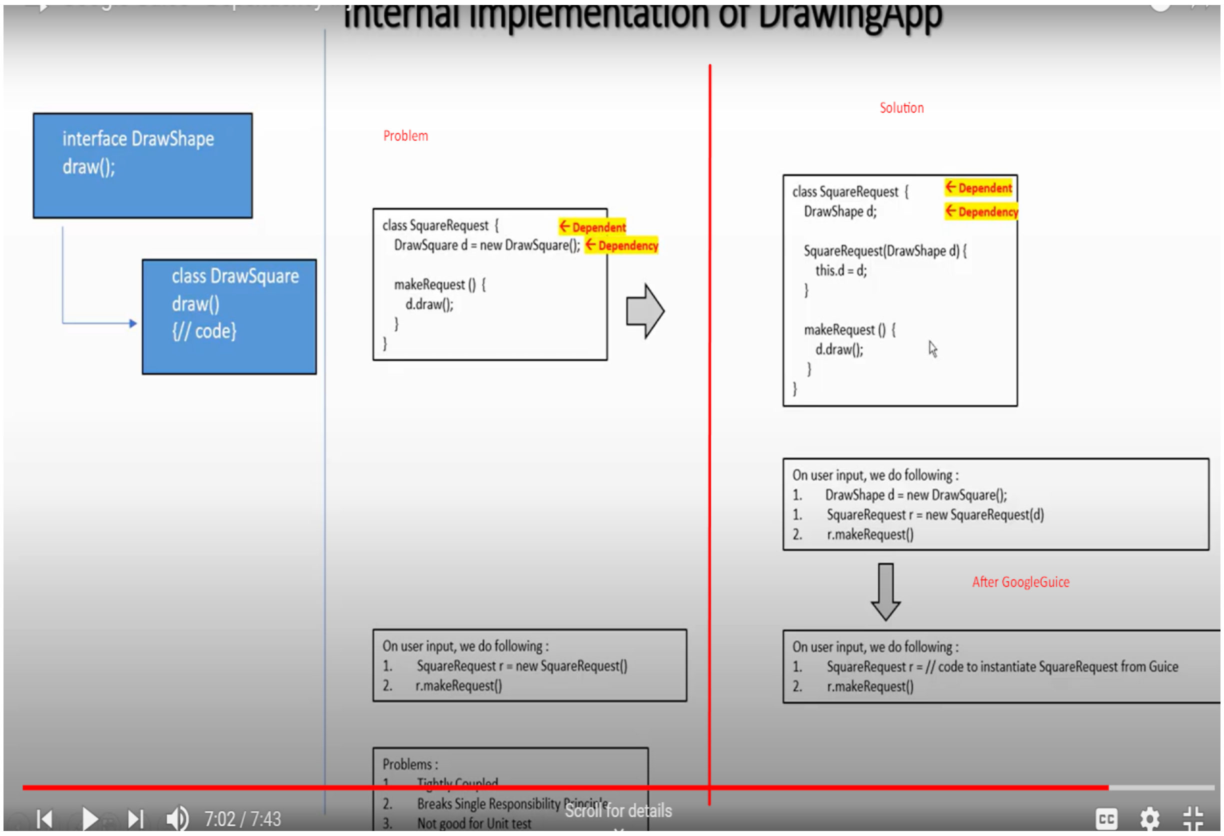
Refer: https://www.youtube.com/playlist?list=PLp0ed20U4R4jknb4xYdhx3yJn5RhWECxn



Internal implementation of DrawingApp

```
interface DrawShape
draw();
```

```
class DrawSquare
draw()
{// code}
```

Problem

```
class SquareRequest {                              ← Dependent
    DrawSquare d = new DrawSquare();               ← Dependency

    makeRequest () {
       d.draw();
    }
}
```

```
On user input, we do following :
1.    SquareRequest r = new SquareRequest()
2.    r.makeRequest()
```

Problems :
1. Tightly Coupled
2. Breaks Single Responsibility Principle
3. Not good for Unit test

Solution

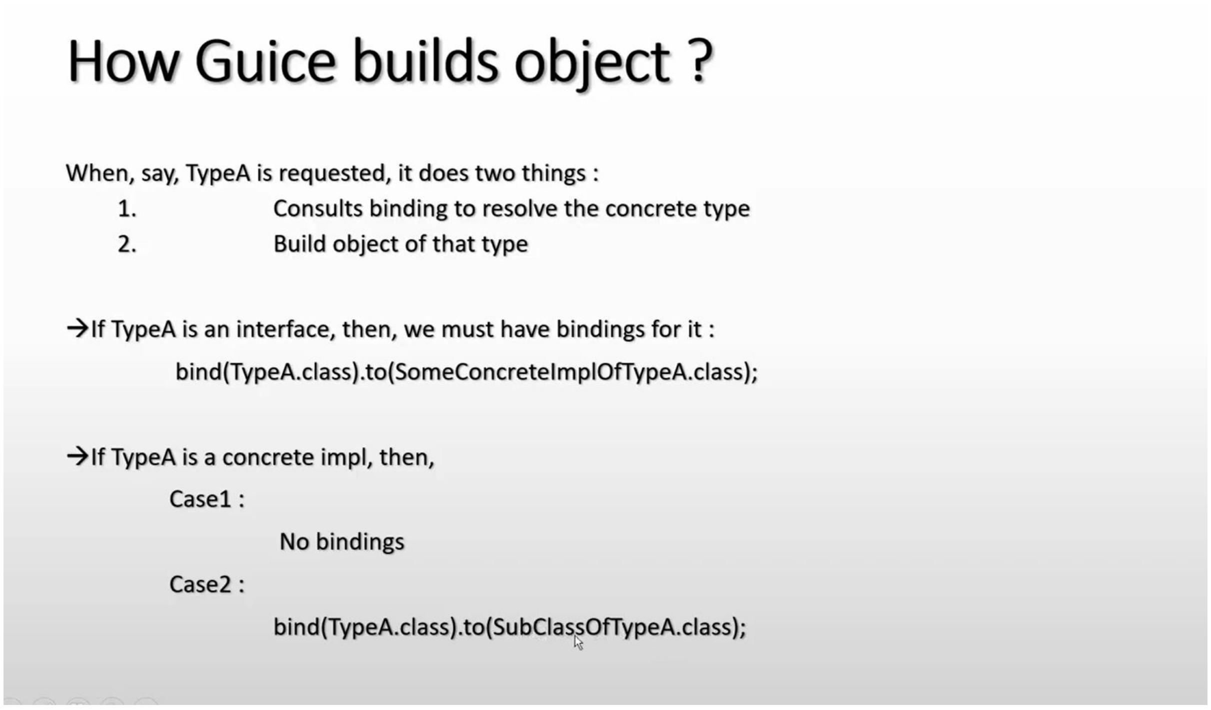
```
class SquareRequest {        ← Dependent
    DrawShape d;             ← Dependency

    SquareRequest(DrawShape d) {
       this.d = d;
    }

    makeRequest () {
       d.draw();
    }
}
```

```
On user input, we do following :
1.    DrawShape d = new DrawSquare();
1.    SquareRequest r = new SquareRequest(d)
2.    r.makeRequest()
```

After GoogleGuice

```
On user input, we do following :
1.    SquareRequest r = // code to instantiate SquareRequest from Guice
2.    r.makeRequest()
```

7:02 / 7:43
Scroll for details



How Guice builds object ?

When, say, TypeA is requested, it does two things :
1. Consults binding to resolve the concrete type
2. Build object of that type

→If TypeA is an interface, then, we must have bindings for it :

```
bind(TypeA.class).to(SomeConcreteImplOfTypeA.class);
```

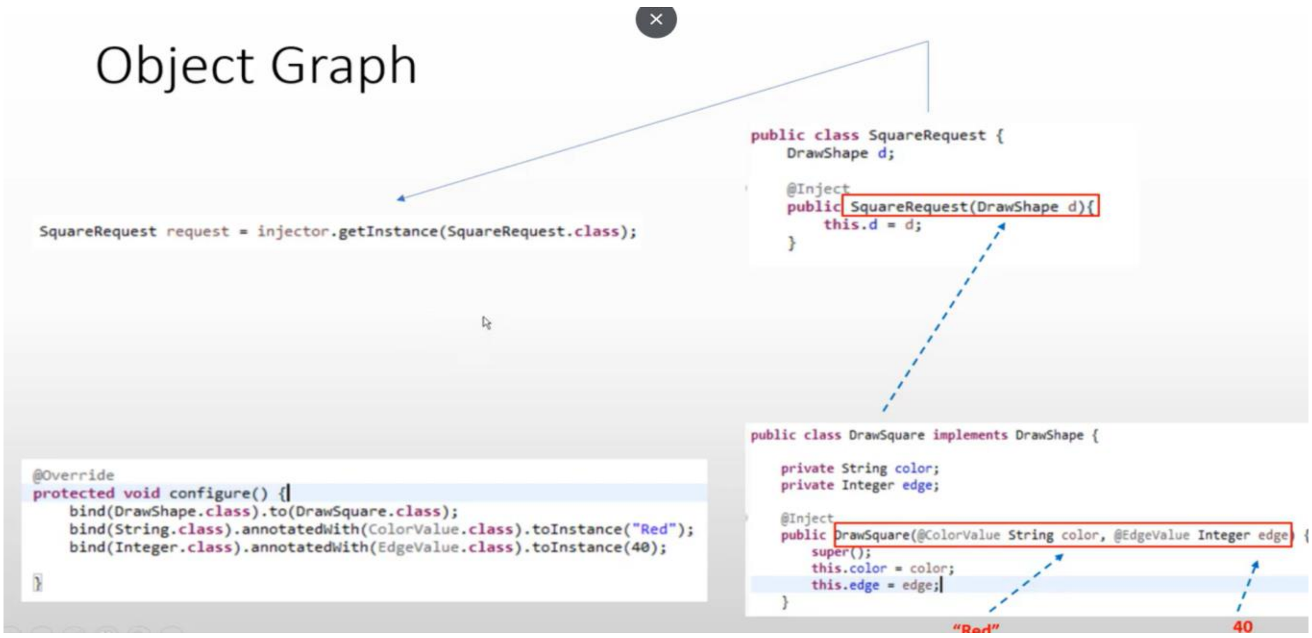
→If TypeA is a concrete impl, then,

Case1 :

No bindings

Case2 :

```
bind(TypeA.class).to(SubClassOfTypeA.class);
```



Object Graph

```
SquareRequest request = injector.getInstance(SquareRequest.class);
```

```
public class SquareRequest {
    DrawShape d;

    @Inject
    public SquareRequest(DrawShape d){
        this.d = d;
    }
```

```
@Override
protected void configure() {
     bind(DrawShape.class).to(DrawSquare.class);
     bind(String.class).annotatedWith(ColorValue.class).toInstance("Red");
     bind(Integer.class).annotatedWith(EdgeValue.class).toInstance(40);

}
```

```
public class DrawSquare implements DrawShape {

    private String color;
    private Integer edge;

    @Inject
    public DrawSquare(@ColorValue String color, @EdgeValue Integer edge) {
        super();
        this.color = color;
        this.edge = edge;
    }
```