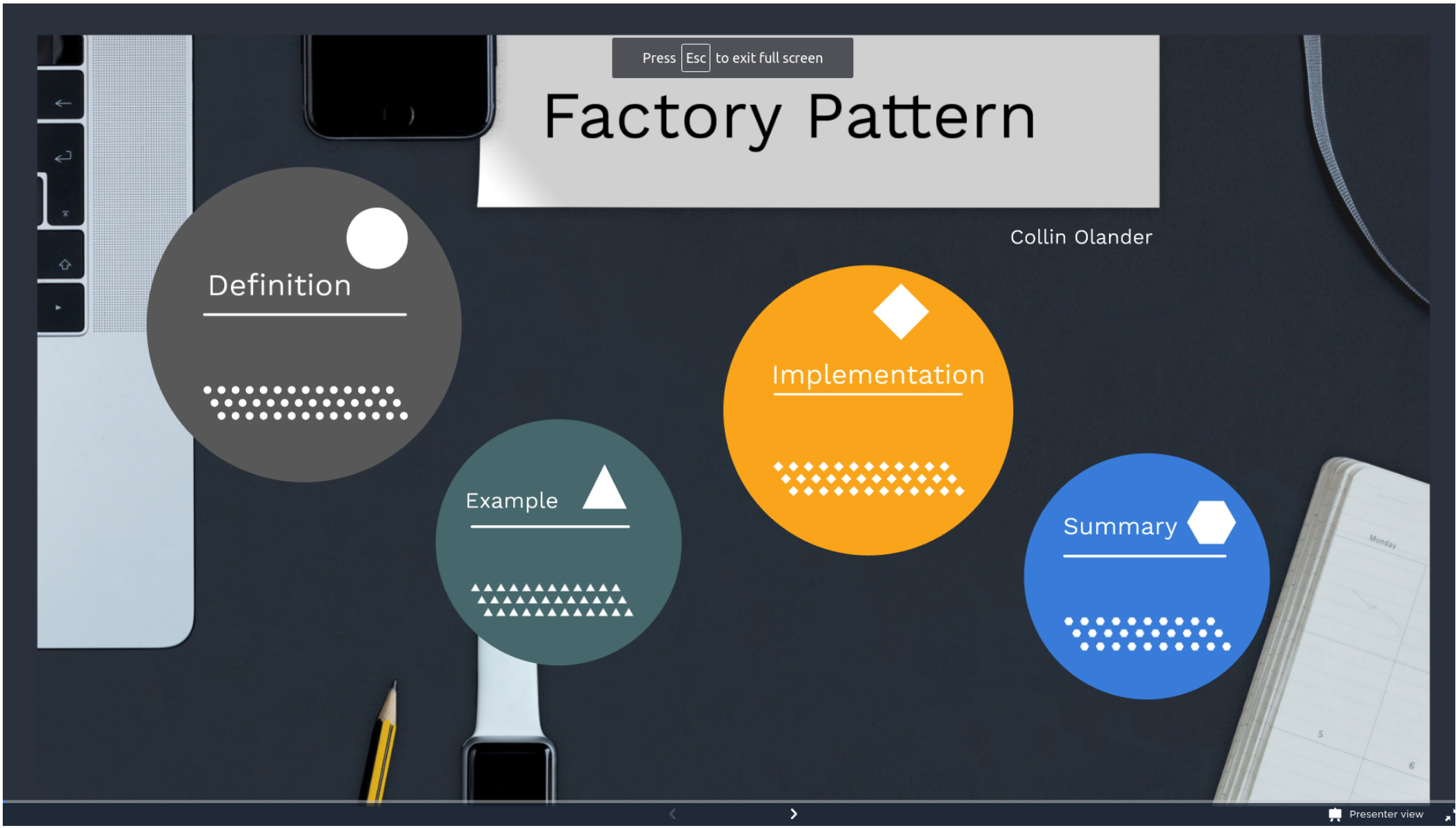
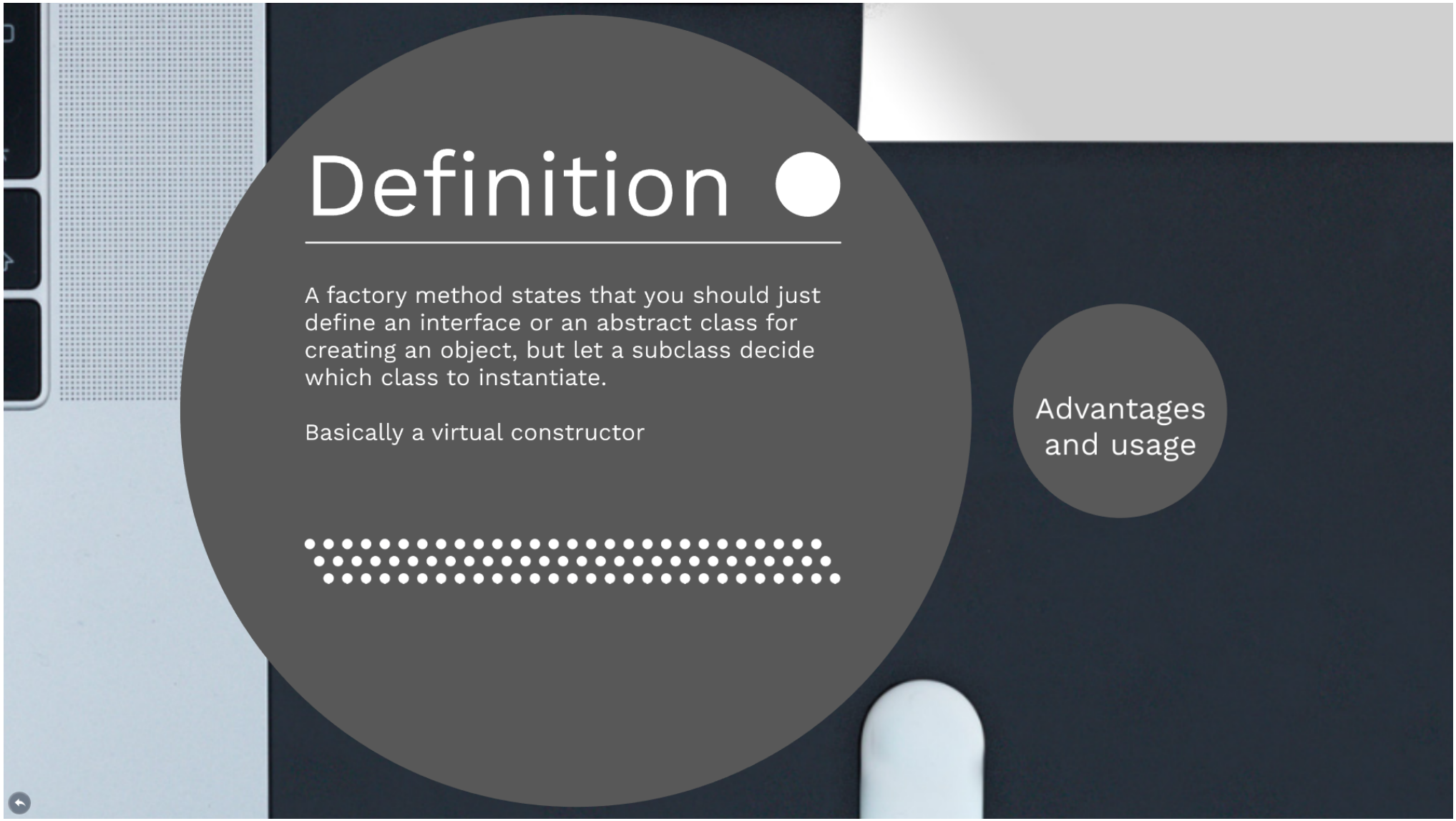

Press Esc to exit full screen
Factory Pattern
Collin Olander
Definition
Example
Implementation
Summary
Presenter view


Definition
A factory method states that you should just define an interface or an abstract class for creating an object, but let a subclass decide which class to instantiate.
Basically a virtual constructor
Advantages and usage

Advantage

Code interaces solely with the
interface or the abstract class.

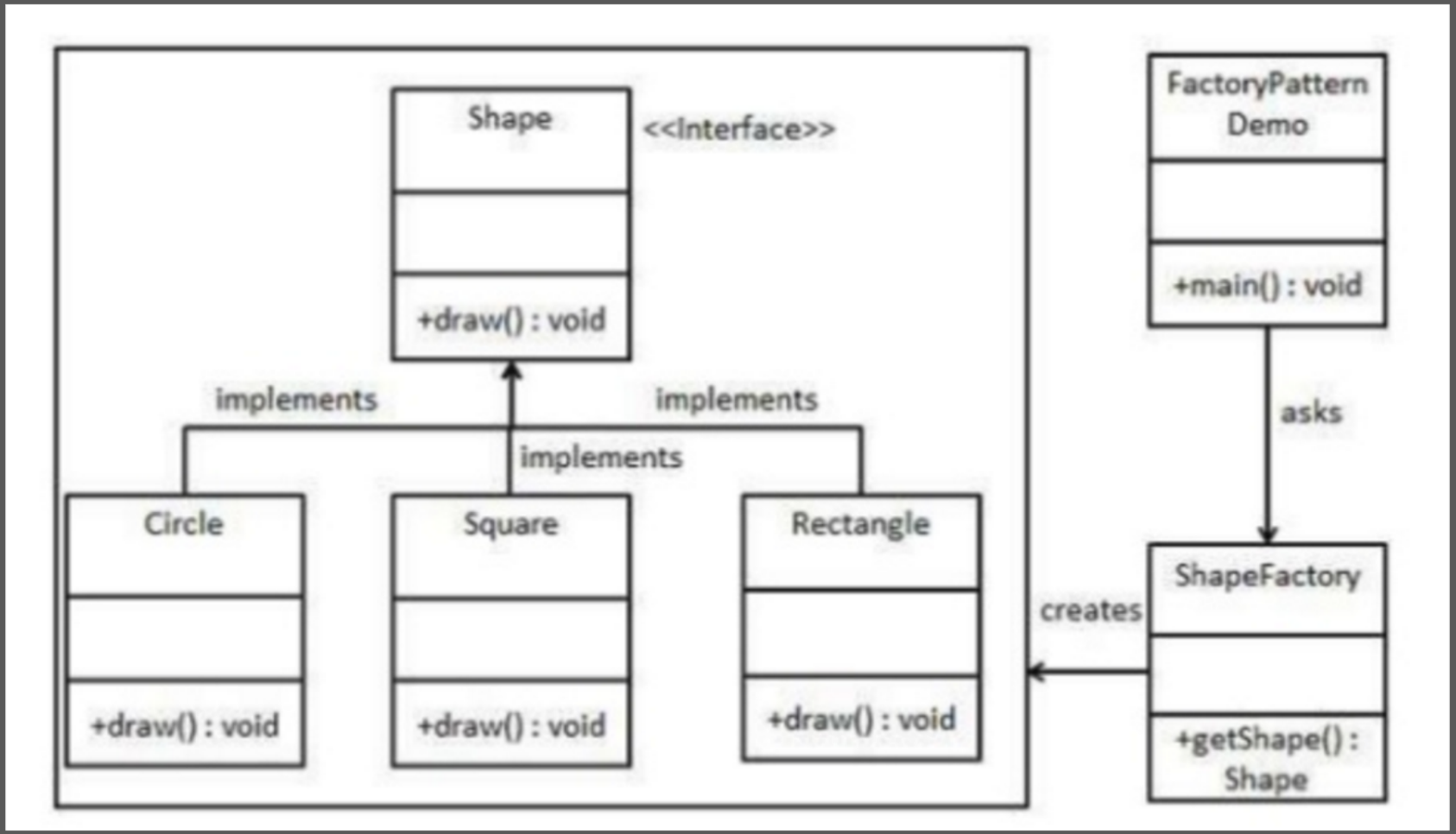
Usage

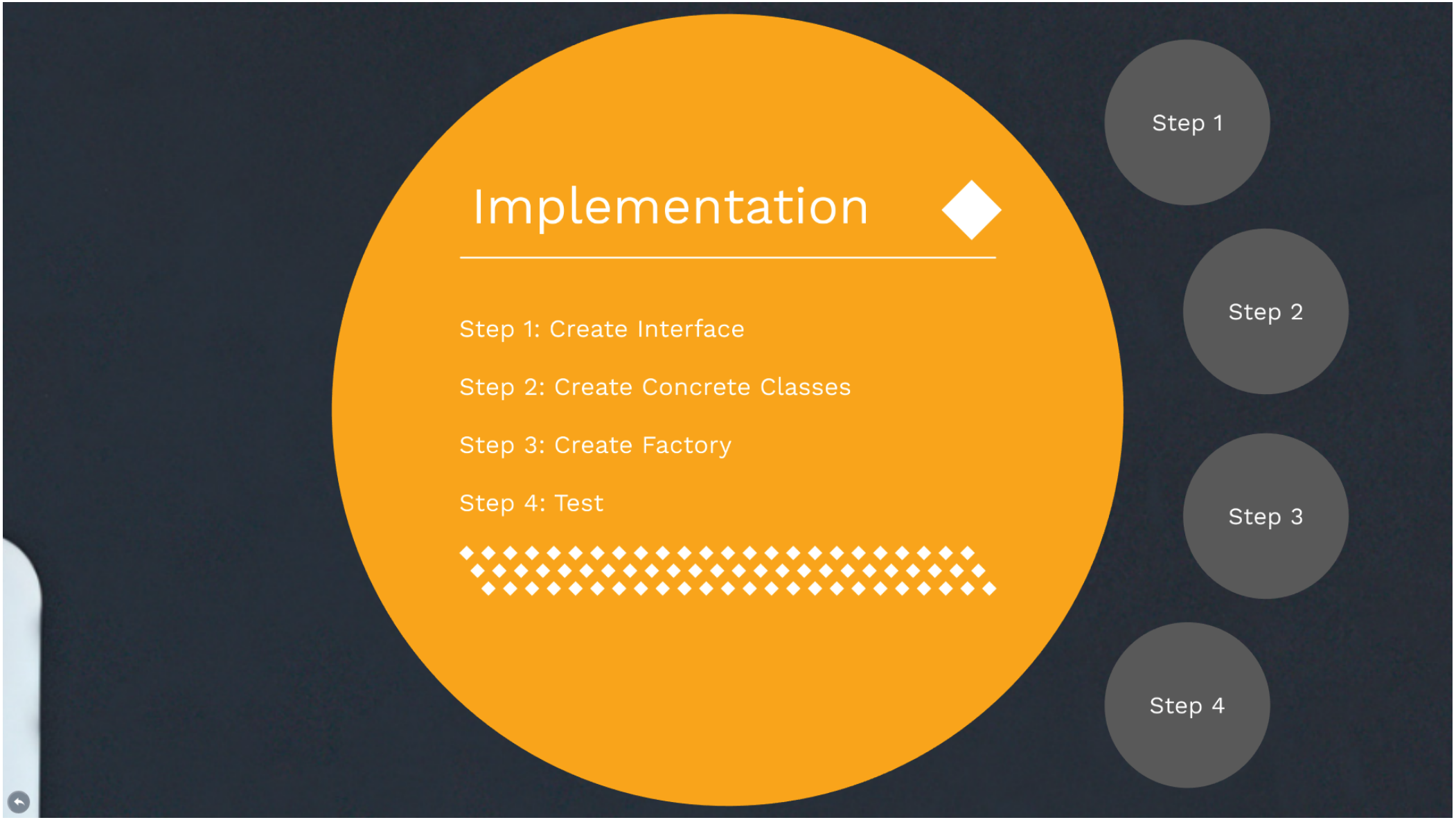
use when:
a class doesn't know what sub-classes will be required
if you want the subclass to specify the objects being created

Example

Let's say we wanted to make a bunch of shapes objects...

UML


Shape
<<Interface>>
+draw() : void
implements
implements
implements
Circle
+draw() : void
Square
+draw() : void
Rectangle
+draw() : void
FactoryPattern Demo
+main() : void
asks
creates
ShapeFactory
+getShape() : Shape


Implementation
Step 1: Create Interface
Step 2: Create Concrete Classes
Step 3: Create Factory
Step 4: Test
Step 1
Step 2
Step 3
Step 4

Step 1: Create Interface

```
1
2  public interface Shape{
3      void draw();
4      void howManySides();
5  }
```

Step 2: Concrete Classes

```java
public class Rectangle implements Shape{
    @Override
    public void draw(){
        System.out.println("Drawing a Reactangle");
    }
    @Override
    public void howManySides(){
        System.out.println("A Rectangle has four sides\nMaybe it's also a square?");
    }
}
```

```java
public class Square implements Shape{
    @Override
    public void draw(){
        System.out.println("Drawing a Square");
    }
    @Override
    public void howManySides(){
        System.out.println("A Square has four sides\nA square is also a rectangle");
    }
}
```

```java
public class Circle implements Shape{
    @Override
    public void draw(){
        System.out.println("Drawing a Circle");
    }
    @Override
    public void howManySides(){
        System.out.println("A Circle has 0 sides... or maybe infinity sides?\nIt's definitely not a square");
    }
}
```

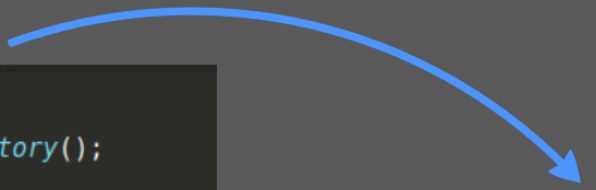
Step 3: The Factory

The factory is what returns the specific object you're looking for

```
1  public class ShapeFactory{
2      public Shape getShape(String shapeType){
3          if (shapeType == null) {
4              return null;
5          }
6          switch(shapeType.toLowerCase()){
7              case "circle":
8                  return new Circle();
9              case "rectangle":
10                 return new Rectangle();
11             case "square":
12                 return new Square();
13         }
14
15         return null;
16     }
17 }
```

Step 4: Test

```
1  public class FactoryPatternApp{
2      public static void main(String[] args) {
3          ShapeFactory shapeFactory = new ShapeFactory();
4
5          Shape shape1 = shapeFactory.getShape("Circle");
6          shape1.draw();
7          shape1.howManySides();
8
9          Shape shape2 = shapeFactory.getShape("Rectangle");
10         shape2.draw();
11         shape2.howManySides();
12
13         Shape shape3 = shapeFactory.getShape("Square");
14         shape3.draw();
15         shape3.howManySides();
16
17
18     }
19 }
```



```
collinol@collin-HP:~/OOArch/Presentation$ java FactoryPatternApp
Drawing a Circle
A Circle has 0 sides... or maybe infinity sides?
It's definitely not a square
Drawing a Reactangle
A Rectangle has four sides
Maybe it's also a square?
Drawing a Square
A Square has four sides
A square is also a rectangle
```

We only need one "Shape" type and we can let the factory build specific shapes for us.

Eliminates the need to rewrite a lot of the same code for multiple shapes.

Summmary

Objective of factory method: create objects that derive from a particular base class

Use when you find yourself rewriting a lot of the same code for related classes or subclasses.

Other examples: Animals, different models of the same brand, etc..