



Bringing British
Education to You
www.nccedu.com

Analysis, Design and Implementation

Topic 7:
Design Patterns (1)

V1.0 © NCC Education Limited

Design Patterns (1) Topic 7 - 7.2

Introduction

- We have spent the past few lectures discussing primarily the analysis and design elements of building software.
- In this lecture, we are going to explore some of the ways we can make our implementation more effective by making use of design patterns.
- Design patterns are particular software development techniques that can be used to achieve design goals.

V1.0 © NCC Education Limited

Design Patterns (1) Topic 7 - 7.3

Design Patterns - 1

- As part the design process of building a piece of software, we encounter situations that we think will be difficult to write in code.
 - Any time you think to yourself 'I am not sure how this is best done'
- Design patterns are collections of objects and classes that are aimed at meeting a particular design need.
 - There are many dozens of well established patterns, and we can only discuss a few of these.

V1.0 © NCC Education Limited

Design Patterns (1) Topic 7 - 7.4

Design Patterns - 2

- Design patterns fill the role of 'high level design' in object-oriented programs.
 - It is often difficult to develop 'good' object-oriented solutions.
 - Design patterns are a shorthand for particular collections of objects, arranged in a particular way.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.5

Design Patterns - 3


- Design patterns are **battle-tested**.
 - A pattern only enters into common acceptance when it has been shown to work in many situations – as such, we can be confident that it works.
- Design patterns are **generalised**.
 - A design pattern is to object design what an algorithm is to lines of code.
 - We need to write it with our specific context in mind.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.6

Representing Design Patterns


- A design pattern is a collection of (usually) several different classes.
- It defines the interaction of these objects at a higher level of abstraction. The examples we see are just examples:
 - You need to implement them in the context of your own systems.
- They are not solutions themselves, just the 'shape' of solutions.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.7

Benefits of Patterns - 1


- Part of what design patterns bring to the field of software engineering is a common vocabulary for design, i.e. shorthand solutions for certain kinds of recurring programming problems.
- Design patterns are 'best practice', and thus allow for portability of the 'lessons learned' of experienced developers.
 - They are an aid to learning - you get the benefit of avoiding the mistakes others have made over the years.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.8

Benefits of Patterns - 2


- They fit in easily with existing modern programming techniques.
 - Design patterns complement object-orientation
 - They are easily expressed as UML diagrams for the most part.
- Design patterns reduce the need for future large-scale refactoring.
 - They are tried and tested, and the result of successive refactoring of their own.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.9

Criticisms of Patterns - 1


- Some of the patterns that are in common usage are simple workarounds for missing language features.
 - Some patterns are repetitions of things supported syntactically in other languages.
- Can lead to systems that are heavily over-engineered.
 - Use with care – it's tempting to put patterns in when there is no real requirement.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.10

Criticisms of Patterns - 2


- There is a saying in English: “When all you have is a hammer, every problem looks like a nail.”
 - This is the tendency of people to assume that the one solution they have available is the one that is always right.
 - Patterns represent good solutions to recurring problems, but they are no substitute for effective analysis.
- Counter-productive when building expertise.
 - You learn more from failure than you do from success.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.11

Design Patterns - 4


- Design patterns are broken down into a number of different categories.
 - Structural
 - Creational
 - Behavioural
- The first of the patterns that we are going to look at is called the **factory** design pattern.
 - This is a creational pattern.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.12

Creational Design Patterns

- We use creational design patterns for several reasons:
 - Some situations are more complex than simple instantiation can handle. Imagine for example you want to create an entirely 'skinnable' look and feel for an application.
 - Some situations have complex consequences if objects aren't instantiated in the right way or the right order.
 - Some situations require that only one object is ever created.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.13

The Factory Pattern

- The **Factory Pattern** is used to provide a consistent interface to setup properly configured objects.
 - You pass in some configuration details...
 - Out comes a properly configured object.
- At its simplest, it can be represented by a single class containing a single static method.
 - More complex factories exist, dealing with more complex situations. These are all variations on the basis structure we will discuss in the lecture.

V1.0

NCC

Bringing British Education to You

www.nccedu.com

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.14

The Factory Design Pattern - 1

- Imagine a class:

Shape

-x: int

-y: int

-length: int

-height: int

+getX(): int

+setX(value: int)

+getY(): int

+setY(value: int)

+setLength(value: int)

+getLength(): int

+getHeight(): int

+setHeight(value: int)

+drawShape(g: Graphics)

V1.0

NCC

Bringing British Education to You

www.nccedu.com

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.15

The Factory Design Pattern - 2

- Then imagine a simple class hierarchy:

```
classDiagram
    class Shape {
        -x: int
        -y: int
        -length: int
        -height: int
        +getX(): int
        +setX(value: int)
        +getY(): int
        +setY(value: int)
        +setLength(value: int)
        +getLength(): int
        +getHeight(): int
        +setHeight(value: int)
        +drawShape(g: Graphics)
    }
    class Circle
    class Rectangle
    class Triangle
    Shape <|-- Circle
    Shape <|-- Rectangle
    Shape <|-- Triangle
```

V1.0

NCC

Bringing British Education to You


www.nccedu.com

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.16

The Factory Design Pattern - 3

- Now imagine you are creating a simple drawing package that works as follows:
 - User selects a shape
 - User clicks on the screen
 - Application draws the shape at that location.
- This can all be hard-coded directly into an application. This suffers from scale and readability issues as the application becomes a maze of 'if' statements and switches.

Bringing British Education to You
www.nccedu.com


V1.0

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.17

The Factory Design Pattern - 4

- Instead, we can use a factory to generate specific objects, through the power of **polymorphism**.
 - Polymorphism is key to the way a factory works.
- The system that drives a factory is that all these shapes have a common parent class. Thus, all we need is the Shape object that is represented by specific objects.
- The objects themselves manage the complexity of the drawing process.

Bringing British Education to You
www.nccedu.com

V1.0

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.18

The Factory Design Pattern - 5


```
public class ShapeFactory {
    public Shape getShape (String shape, int x, int y, int len, int ht,
    Color col) {
        Shape temp = null;

        if (shape.equals ("Circle")) {
            temp = new Circle (x, y, len, ht);
        }

        else if (shape.equals ("Rectangle")) {
            temp = new Rectangle (x, y, len, ht);
        }

        else if (shape.equals ("Face")) {
            temp = new Face (x, y, len, ht);
        }

        temp.setDrawingColor (col);
        return temp;
    }
}
```

Bringing British Education to You
www.nccedu.com


V1.0

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.19

The Factory Design Pattern - 6


- The Factory Pattern reduces hard-coded complexity. We don't need to worry about combinatorial explosion.
- The Factory Pattern properly devolves responsibility to individual objects.
 - We don't have a draw method in our application, we have a draw method in each specific shape.
- However, the Factory Pattern by itself is limited to certain simple contexts. For more complicated situations, we need more.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.20

The Abstract Factory - 1


- The next level of abstraction is the **Abstract Factory**.
 - This is a factory *for* factories.
- Imagine that we have slightly more complicated situations:
 - Designing an interface that allows for different themes.
 - A file conversion application that must allow for different versions of different formats.
 - A bank that provides different kinds of accounts for their customers.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.21

The Abstract Factory - 2


- We could handle these with a factory by itself.
 - But this introduces the same combinatorial problems that the factory is designed to resolve.
- A simple rule to remember is – coding combinations is usually bad design.
- Bad design causes trouble later on.
- When doing anything more substantial than simple 'proof of concept' applications, we should always be sure to engineer our systems properly.

V1.0  Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.22

Bad Design

```
public Component getComponent (String type, String theme) {  
    Component guiComponent;  
  
    if (theme.equals ("swing")) {  
        if (type.equals ("button")) {  
            guiComponent = new JButton ();  
        }  
        else if (type.equals ("label")) {  
            guiComponent = new JLabel();  
        }  
    }  
    else if (theme.equals ("awt")) {  
        if (type.equals ("button")) {  
            guiComponent = new Button ();  
        }  
        else if (type.equals ("label")) {  
            guiComponent = new Label();  
        }  
    }  
    return guiComponent;  
}
```


Bringing British
Education to You
www.nccedu.com

V1.0 © NCC Education Limited

Design Patterns (1) Topic 7 - 7.23

Good Design - 1

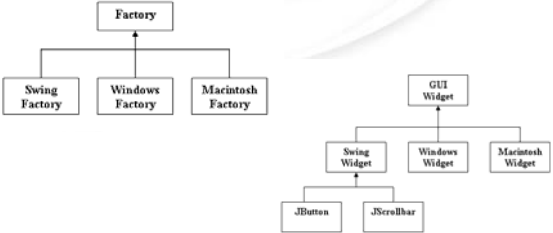
- Good Design in this case involves creating one factory that creates the right kind of factory for the components:
 - We have a SwingFactory and an AWTFactory.
 - That factory generates the appropriate components.
- This requires a somewhat more complicated class structure.
 - Each factory must inherit from a common base.
 - We get the factory and then use that to create our objects.

Bringing British
Education to You
www.nccedu.com


V1.0 © NCC Education Limited

Design Patterns (1) Topic 7 - 7.24

Good Design - 2



```
graph TD  
    Factory[Factory] --> SwingFactory[Swing Factory]  
    Factory --> WindowsFactory[Windows Factory]  
    Factory --> MacintoshFactory[Macintosh Factory]  
    GUIWidget[GUI Widget] --> SwingWidget[Swing Widget]  
    GUIWidget --> WindowsWidget[Windows Widget]  
    GUIWidget --> MacintoshWidget[Macintosh Widget]  
    SwingWidget --> JButton[JButton]  
    SwingWidget --> JScrollbar[JScrollbar]
```

Bringing British
Education to You
www.nccedu.com

V1.0 © NCC Education Limited


Design Patterns (1) Topic 7 - 7.25

Abstract Factory Implementation

```
public class AbstractFactory {
    public static Factory getFactory (string look) {
        Factory temp;
        if (look.equals ("windows")) {
            temp = new WindowsFactory();
        }

        else if (look.equals ("swing")) {
            temp = new SwingFactory();
        }

        else if (look.equals ("macintosh")) {
            temp = new MacintoshFactory();
        }
        return temp;
    }
}
```

Bringing British
Education to You
www.nccedu.com

V1.0

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.26

Factory Implementation


```
public class SwingFactory extends Factory {

    public GuiWidget getWidget (string type) {
        SwingWidget temp = null;
        if (type.equals ("button")) {
            temp = new JButton();
        }

        else if (type.equals ("scrollbar")) {
            temp = new JScrollbar();
        }

        return temp;
    }

    abstract class Factory {
        abstract GuiWidget getWidget (String type);
    }
}
```

Bringing British
Education to You
www.nccedu.com

V1.0

© NCC Education Limited


Design Patterns (1) Topic 7 - 7.27

The Consequence

- Entirely new suites of themes can be added to this system without risking combinatorial explosion.
- The 'operational' code is also much tighter and more focused.

```
Factory myFactory =
AbstractFactory.getFactory ("swing");

GuiWidget myWidget = myFactory.getWidget
("button");
```

Bringing British
Education to You
www.nccedu.com

V1.0

© NCC Education Limited

Design Patterns (1) Topic 7 - 7.28

Using Patterns - 1

- Patterns are best decided upon in the design phase of development.
 - During our analysis, we may encounter situations that require the creation of large amounts of objects.
- We use the factory and/or abstract factory when we must deal with potentially large combinations of objects that must be created and configured.
- If we see such a requirement, we make sure to place our design pattern in the appropriate part of our class diagram.

V1.0 NCC Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.29

Using Patterns - 2

- Much of deciding when we should use patterns is based on our own pattern recognition.
 - We see that a particular kind of functionality is going to be suited to a pattern of which we are aware.
- Experimenting with patterns is an important part of building that skill.
- We have to see what patterns can do, how they do it, and how they can be moulded to meet our own specific needs.

V1.0 NCC Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.30

Conclusion

- Design patterns are an important technique for handling the implementation of complex functionality.
 - They can be described as algorithms for object and class design.
- The Factory Design Pattern is used to handle the creation of objects that stem from a common base.
- Abstract factories are factories *for* factories.
 - We choose the factory we want, and use that to generate objects.

V1.0 NCC Bringing British Education to You www.nccedu.com © NCC Education Limited

Design Patterns (1) Topic 7 - 7.31

Topic 7 – Design Patterns (1)

Any Questions?



Bringing British
Education to You
www.nccedu.com



V1.0

© NCC Education Limited
