

COMP50002 Software Engineering Design

Imperial College London

Boxuan Tang

Autumn 2022

Contents

1	Mock Objects	2
2	Designing for Flexibility	2
3	Re-use and Extensibility	2
3.1	Template Method Pattern	2
3.2	Strategy Pattern	2
4	Creation and Dependency	3
4.1	Factory Method	3
4.2	Factory Object	3
4.3	Builder	3
4.4	Singleton	3
5	Concurrency	3
6	Interactive Applications	4
7	System Integration	5
7.1	Adapter	5
7.2	Decorator	5
7.3	Proxy	5

1 Mock Objects

```
public class TestHeadChef {
    @Rule public JUnitRuleMockery context = new JUnitRuleMockery();

    Chef pastryChef = context.mock(Chef.class);
    HeadChef headChef = new HeadChef(pastryChef);

    @Test public void delegatesPuddingsToPastryChef() {
        context.checking(new Expectations({{
            exactly(1).of(pastryChef).order("Pudding"); // ignoring() allowing() never()
            will(returnValue("Ordered Pudding"));
        }}));

        headChef.order("Chicken", "Pudding");
    }
}
```

2 Designing for Flexibility

Definition 2.1 (Law of Demeter) *Each unit should only talk to its immediate friends*

Definition 2.2 (Tell, Don't Ask Style) *Objects send messages to one another in order to pass information and get things done → Otherwise fragile + tightly coupled*

3 Re-use and Extensibility

3.1 Template Method Pattern

```
public abstract class NumberSequence {
    public int term(int i) {
        if (i < 0) throw new IllegalArgumentException("Negative Undefined");
        return positiveTerm(i);
    }

    protected abstract int positiveTerm(int i);
}

public class TriangleNumberSequence extends NumberSequence {
    @Override protected int positiveTerm(int i) { return (i+1)*(i+2)/2 }
}
```

3.2 Strategy Pattern

```
public class NumberSequence {
    private final TermGenerator termGenerator;
    public NumberSequence(TermGenerator termGenerator) {this.termGenerator = termGenerator;}

    public int term(int i) {
        if (i < 0) throw new IllegalArgumentException("Negative Undefined");
        return termGenerator.positiveTerm(i);
    }
}

public class TriangleNumberSequence implements TermGenerator {
    @Override public int positiveTerm(int i) { return (i+1)*(i+2)/2 }
}
```

4 Creation and Dependency

4.1 Factory Method

```
class VirtualMachines {
    private final int memory;

    public static VirtualMachine highMemory() {return new VirtualMemory(100);}
    private VirtualMachine(int memory) {this.memory = memory;} // forces factory methods
}
```

4.2 Factory Object

```
class LogoFactory implements Supplier<Logo> {
    static Logo get() {
        if (config.country().equals(Country.UK)) return new FlagLogo("Union Jack");
        return new DefaultLogo();
    }
}
```

4.3 Builder

```
public class BookSearchQueryBuilder {
    private String firstName = null;
    private BookSearchQueryBuilder() {}

    public static BookSearchQueryBuilder books() {return new BookSearchQueryBuilder();}
    public BookSearchQueryBuilder withFirstName(String firstName) {
        this.firstName = firstName; return this;
    }
    public build() {return new BookSearchQuery(firstName);}
}
```

4.4 Singleton

```
public class LibraryCatalogue {
    private static LibraryCatalogue instance = new LibraryCatalogue();
    private LibraryCatalogue() {}
    public static LibraryCatalogue getInstance() {return instance;}
}
```

5 Concurrency

```
.. public static void main(String[] args) {
    ExecutorService executorService = Executors.newFixedThreadPool(2);
    Future<Double> future = executorService.submit(myCallable);

    executorService.shutdown();
    try { executorService.awaitTermination(120, TimeUnit.Seconds);
    } catch (InterruptedException e) {...}

    Double result = future.get();
}

public class LatchedTask implements Runnable {
    private final CountDownLatch latch; private final Runnable task;

    public LatchedTask(Runnable task, CountDownLatch latch) {...}
}
```

```

    @Override public void run() {task.run(); latch.countdown();}
}

.. public static void main(String[] args) {
    CountdownLatch latch = new CountdownLatch();
    executorService.submit(new LatchedTask(myRunnable, latch));
    try {latch.await();
    } catch (InterruptedException e) {...}
}

```

6 Interactive Applications

```

public class Calculator implements Updatable {
    private final JTextField output = new JTextField(10);
    // view
    private void display() {
        JFrame frame = new JFrame("Calculator");
        ArithmeticEngine calc = new ArithmeticEngine();
        calc.addObserver(this);
        frame.setSize(350,300);
        JPanel panel = new JPanel();

        panel.add(output);
        addNumberButtons(panel, calc); addOperatorButtons(panel, calc);

        frame.getContentPane().add(panel);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    @Override public void updateWith(int value) {
        output.setText(String.valueOf(value));
    }

    // controller
    private addNumberButtons(JPanel panel, ArithmeticEngine calc) {
        IntStream.range(1,5).forEach(n->{
            JButton button = new JButton(String.valueOf(n));
            button.addActionListener(e->calc.input(n));
            panel.add(button);
        });
    }

    private addOperatorButtons(JPanel panel, ArithmeticEngine calc) {
        EnumSet.allOf(Operator.class).forEach(op->{
            JButton button = new JButton(op.label());
            button.addActionListener(e->calc.apply(op));
            panel.add(button);
        });
    }

    public static void main(String[] args) {new Calculator().display();}
}

enum Operator {
    PLUS("+") {@Override public Integer apply(Integer x, Integer y) {return y + x;}},

```

```

    MINUS("-") {@Override public Integer apply(Integer x, Integer y) {return y - x;}};

    private String label;
    Operator(String label) {this.label = label;}
    public String label() {return label;}
    public abstract Integer apply(Integer x, Integer y);
}

public class ArithmeticEngine { // model
    private final List<Updatable> = new ArrayList();
    private final Stack<Integer> = new Stack();

    public void input(int value) {stack.push(value); notifyObservers();}
    public void apply(Operator op) {
        stack.push(op.apply(stack.pop(), stack.pop())); notifyObservers();
    }

    public void addObservers(Updatable observer) {observers.add(observer);}

    private void notifyObservers() { // model should update view directly
        for (Updatable observer : observers) {observer.updateWith(stack.peek());}
    }
}

```

7 System Integration

Hexagonal Architectures

7.1 Adapter

Convert the interface of a class into another interface clients expect.

```

public class WeatherDotComTemperatureService implements TemperatureService {
    Forecaster forecaster = new Forecaster();
    @Override // throws IllegalArgumentException
    public int temperatureFor(String place, DayOfWeek day) {
        return forecaster.forecastFor(com.weather.Region.valueOf(place.toUpperCase()),
            com.weather.Day.valueOf(day.name().toUpperCase()).temperature();
    }
}

```

7.2 Decorator

Add additional functionality or responsibility to an object dynamically.

7.3 Proxy

Control access to an object by providing a placeholder or surrogate object.

```

.. public CachingService(Service downstream, int capacity) {
    this.downstream = downstream; this.cache = new LinkedHashMap<Pair<A,B>,C>() {
        @Override // Pair<A,B> must override equals() and hashCode()
        protected boolean removeEldestEntry(Map.Entry eldest) {return size() > capacity;}
    }
}

```