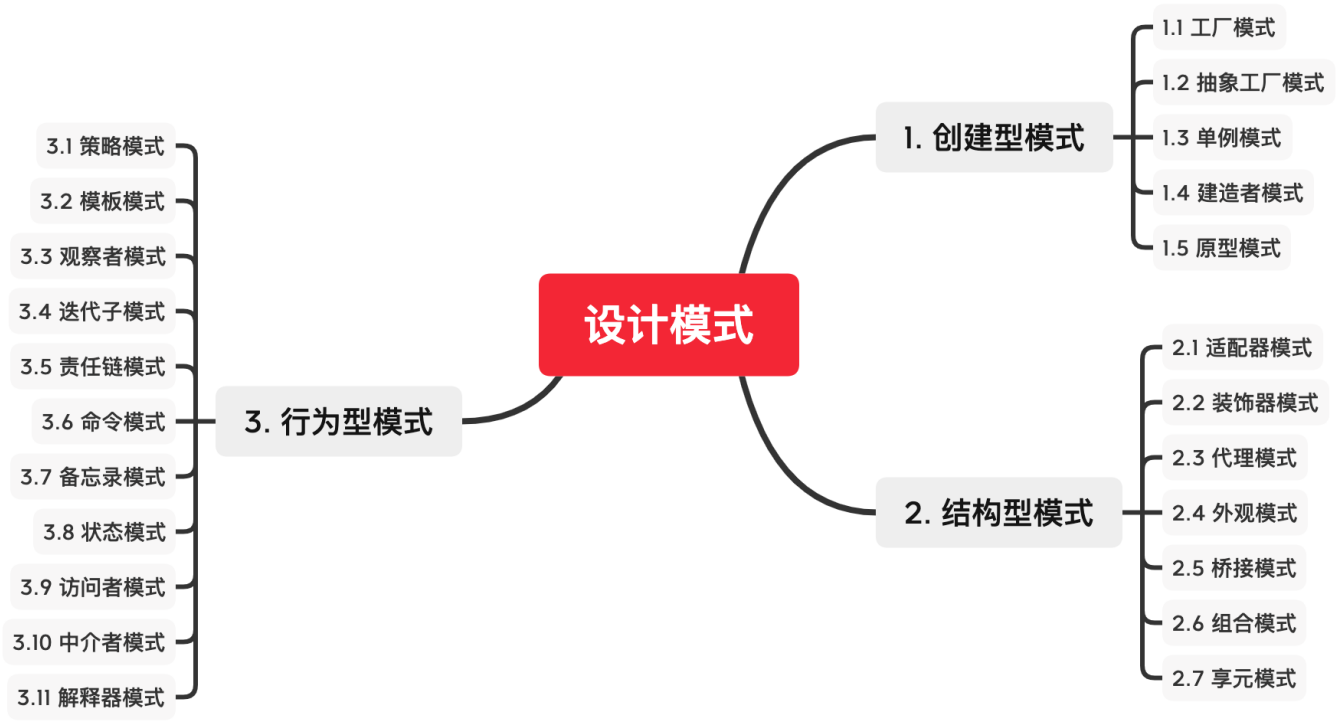
Design Pattern

grayson

2022-07-13

1



1.1

1. ** **
2.
3. ** **
4.
5.

1.2

1.
2. ** **
3. ** **
4.

5.
6.
7.

1.3

1. ** **
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.

1.

1.0

1

1. Singleton Design Pattern
2.
    1.
    2.
    3.
3.
    1.
    2.
    3.

2

2.1

1.
2.
3.

2.2

1.
2.
3.
    1.
    2.

3

1. new
2.
3. **IO**

3.1

1.

2.

3.

```
1. public class Counter {
2. 
3.     private static class CounterHolder{
4.         private static final Counter counter = new Counter();
5.     }
6. 
7.     private Counter(){
8.         System.out.println("init...");
9.     }
10. 
11.     public static final Counter getInstance(){
12.         return CounterHolder.counter;
13.     }
14. 
15.     private AtomicLong online = new AtomicLong();
16. 
17.     public long getOnline(){
18.         return online.get();
19.     }
20. 
21.     public long add(){
22.         return online.incrementAndGet();
23.     }
24. }
```

3.2

1.

2. properties

1. Spring @PropertySource
2. new

3.

```java
public class SingleProperty {
```

```
private static Properties prop;

private static class SinglePropertyHolder{
    private static final SingleProperty singleProperty = new SingleProperty();
}

/**
* config.properties          test.name=kite
*/
private SingleProperty(){
    System.out.println("                  ");
    prop = new Properties();
    InputStream stream = SingleProperty.class.getClassLoader()
            .getResourceAsStream("config.properties");
    try {
        prop.load(new InputStreamReader(stream, "utf-8"));
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static SingleProperty getInstance(){
    return SinglePropertyHolder.singleProperty;
}
```

```
public String getName(){
    return prop.get("test.name").toString();
}

public static void main(String[] args){
    SingleProperty singleProperty = SingleProperty.getInstance();
    System.out.println(singleProperty.getName());
}
```

}

````

### 3.3

1. **           **                                   **           **     **                 **
   **               **             **                   **
     **             **
2.                    **                      **  **             **  **                    **

3.          `Spring`          `Druid`  `C3P0`     **                 **         **             **

## 4

### 4.1

1. **                  **    **          **  **       `instance`             **
2.

```java
public class Singleton {
  private static final Singleton instance = new Singleton();

  private Singleton () {}

  public static Singleton getInstance() {
    return instance;
  }
}
````

3.

    1. instance **Singleton** **<clinit>()**
        1. <clinit>() static static{} private static final Singleton instance = new Singleton();
        2. **<clinit>()** <clinit>() <clinit>()
        3. <clinit>() <clinit>() <clinit>()
    2.

4.

MrDoc
````

1. **instance**

4.2

1.
2.
3.

```
public class Singleton {
  private static final Singleton instance;

  private Singleton () {}

  public static synchronized Singleton getInstance() {
    if (instance == null) {
      instance = new Singleton();
    }

    return instance;
  }
}
```

4.
    1.
    2.
5.
    1.
    2.
    3.

4.3

1.
2. synchronized synchronized

```
public class Singleton {
  private static Singleton instance;

```

```
  private Singleton () {}

public static Singleton getInstance() {
  if (instance == null) {  //
    synchronized(Singleton.class) { //
      if (instance == null) {       //
        instance = new Singleton();
      }
    }
  }
  return instance;
}
}
```

3. Java 1.4

1. instance = new Singleton(); Java

```
// 1
memory = allocate();
// 2
ctorInstance(memory);
// 3          instance
instance = memory;
```

2. 2 3

```
// 1
memory = allocate();
// 3          instance
instance = memory;
// 2
ctorInstance(memory);
```

3.

	A	B
t_1	A_1	
t_2	A_3 $instance$	

	A	B
t_3		B_1 $instance$
t_4		B_2 $instance$ $null$ B $instance$
t_5	A_2	
t_6	A_4 $instance$	

4. B B

5. instance volatile 2.13 Volatile

```
1. public class Singleton {
2.   private static volatile Singleton instance;
3.
4.   private Singleton () {}
5.
6.   public static Singleton getInstance() {
7.     if (instance == null) {  //
8.       synchronized(Singleton.class) { //
9.         if (instance == null) {       //
10.          instance = new Singleton();
11.        }
12.      }
13.    }
14.    return instance;
15.  }
16. }
```

6.

   1.
   2.
   3.

4.4

1. **Java**

   1. **Java**
   2. **JVM**

2.

```
public class Singleton {
  private Singleton () {}

  private static class SingletonInner {
    private static final Singleton instance = new Singleton();
  }

  public static Singleton getInstance() {
    return SingletonInner.instance;
  }
}
```

3. SingletonInner **Singleton** **SingletonInner** **getInstance()** SingletonInner **instance**

4. instance **JVM** 3.4

5.

1.
2.
3.

4.5

1. **Java**

1. enum **final** **Enum** **static** **enum** 3.1

2.

```
public enum T {
   SPRING,SUMMER,AUTUMN,WINTER;
}
```

3.

```
public static final T SPRING;
public static final T SUMMER;
public static final T AUTUMN;
```

```
4. public static final T WINTER;
5. private static final T $VALUES[];
6. static
7. {
8.     SPRING = new T("SPRING", 0);
9.     SUMMER = new T("SUMMER", 1);
10.    AUTUMN = new T("AUTUMN", 2);
11.    WINTER = new T("WINTER", 3);
12.    $VALUES = (new T[] {
13.        SPRING, SUMMER, AUTUMN, WINTER
14.    });
15. }
```

2.

```
1. public enum Singleton {
2.   INSTANCE; //
3. }
```

3.

    1.
    2.
    3.

4.

    1.

5

1.
2.
3. Java
4. 1.3 –
5.
6. Hungry Chinese Style of Singleton Design Pattern.
7. ——
8. -
9. K

1.1

1

1.
2.
3.

2

2.1

1. **new**
2.
3.

3

1. **new**
2.
3.

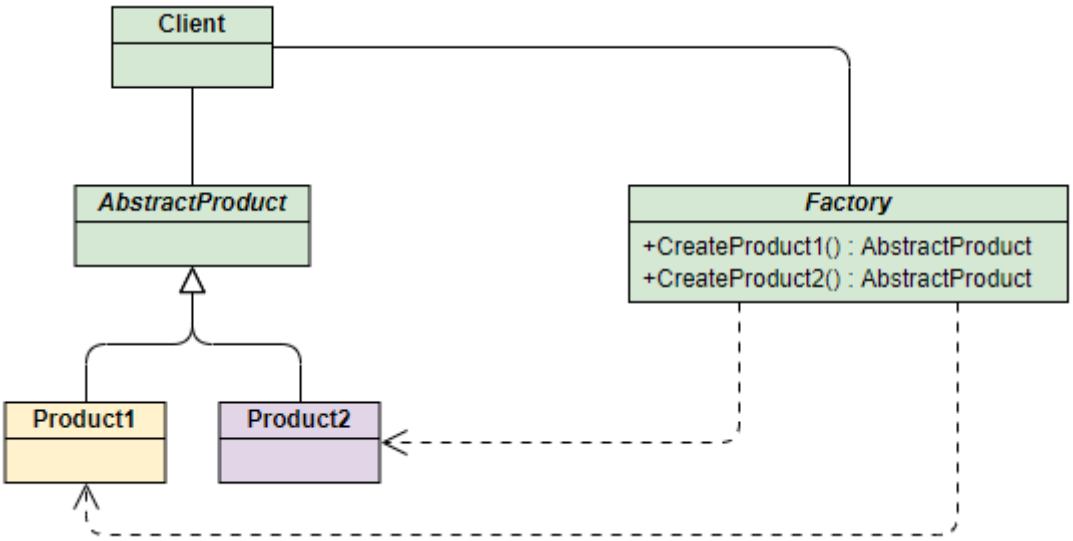
4

4.1

4.1.1

1.
2.

4.1.2 UML



4.1.3

1.

```
1. public interface Car {
2.  void run();
3. }
```

2.

```
1. public class BMW implements Car {
2.     public void run() {
3.         System.out.println("                    ...");
4.     }
5. }
```

3.

```
1. public class AoDi implements Car {
2.     public void run() {
3.         System.out.println("                    ...");
4.     }
5. }
```

4.

```
1. public class CarFactory {
2.     public static Car createCar(String name) {
3.         if (name.equals("       ")) {
```

```
            return new AoDi();
        }
        if (name.equals("      ")) {
            return new BMW();
        }
        return null;
    }
}
```

5.

```
public class FactoryTest {
    public static void main(String[] args) {
        Car aodi = CarFactory.createCar("      ");
        aodi.run();

        Car bmw = CarFactory.createCar("      ");
        bmw.run();
    }
}
```

4.1.4

4.1.4.1

1.

4.1.4.2

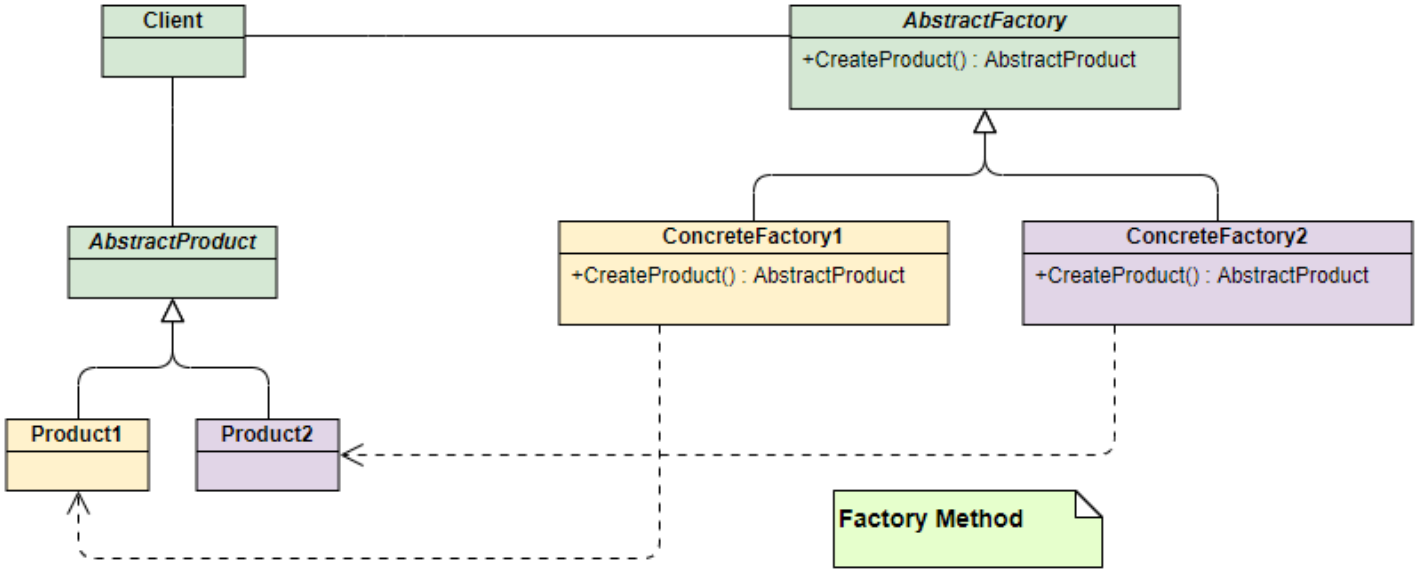
1.

4.2

4.2.1

1.

4.2.2 UML



4.2.3

1.

```
public interface Car {
    void run();
}
```

2. new

```
public interface CarFactory {
    Car createCar();
}
```

3.

```
public class AoDi implements Car {
    public void run() {
        System.out.println("                ...");
    }
}
```

4.

```
public class BMW implements Car {
    public void run() {
        System.out.println("                ...");
    }
```

```
}
```

5.

```
public class AoDiFactory implements CarFactory{
    @Override
    public Car createCar() {
        return new AoDi();
    }
}
```

6.

```
public class BMWFactory implements CarFactory {
    @Override
    public Car createCar() {
        return new BMW();
    }
}
```

7.

```
public class FactoryTest {
    public static void main(String[] args) {
        Car aodi = new AoDiFactory().createCar();
        aodi.run();

        Car bmw = new BMWFactory().createCar();
        bmw.run();
    }
}
```

4.2.4

4.2.4.1

1. ** **
2.

4.2.4.2

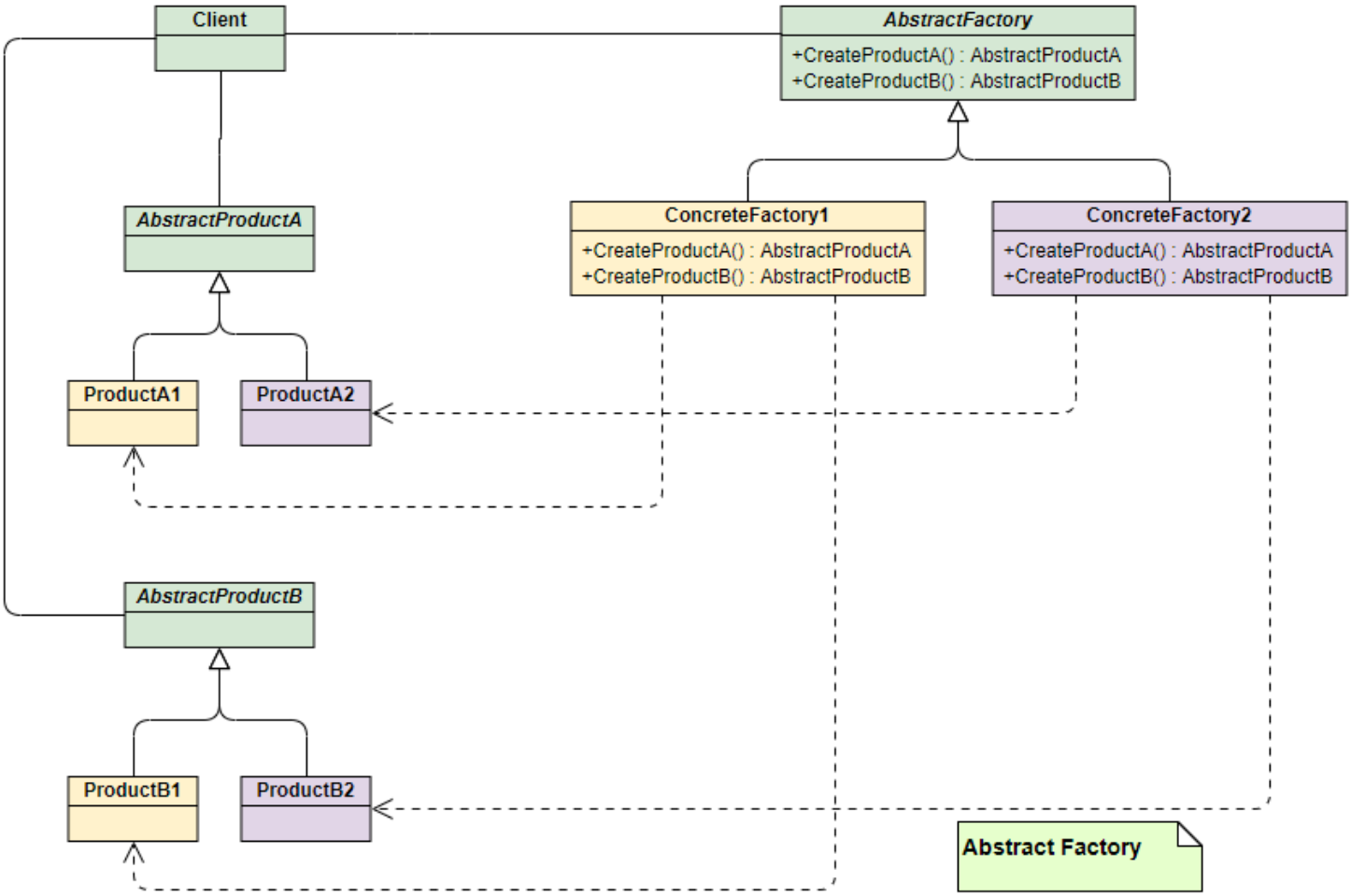
1.

4.3

4.3.1

1.

4.3.2 UML



4.3.3

1.

```java
public interface Car {
    void run();
}

public class AoDi implements Car {
    public void run() {
        System.out.println("                ...");
    }
}

```

```
public class BMW implements Car {
    public void run() {
        System.out.println("                ...");
    }
}
```

2.

```
public interface Engine {
    void spin();
}

public class AoDiEngine implements Engine{
    @Override
    public void spin() {
        System.out.println("        ");
    }
}

public class BMWEngine implements Engine{
    @Override
    public void spin() {
        System.out.println("        ");
    }
}
```

3.

```
public interface AbstractFactory {
    Car createCar();
    Engine createEngine();
}

public class AoDiFactory implements AbstractFactory{
    @Override
    public Car createCar() {
        return new AoDi();
    }

    @Override
    public Engine createEngine() {
        return new AoDiEngine();
    }
```

```
}

public class BMWFactory implements AbstractFactory{
    @Override
    public Car createCar() {
        return new BMW();
    }

    @Override
    public Engine createEngine() {
        return new BMWEngine();
    }
}
```

4.

```
public class FactoryTest {
    public static void main(String[] args) {
        AoDiFactory aoDiFactory = new AoDiFactory();
        Car aodi = aoDiFactory.createCar();
        aodi.run();
        Engine aodiEngine = aoDiFactory.createEngine();
        aodiEngine.spin();

        BMWFactory bmwFactory = new BMWFactory();
        Car bmw = bmwFactory.createCar();
        bmw.run();
        Engine bmwEngine = bmwFactory.createEngine();
        bmwEngine.spin();
    }
}
```

4.3.4

4.3.4.1

1.

2.
3.

4.3.4.2

1.

1.
2. ——
3. factory pattern

2.0

1

1. /
2.

> 1. Proxy
> 2.

2

1.
2.
3.
4.
5. **Spring AOP**

3

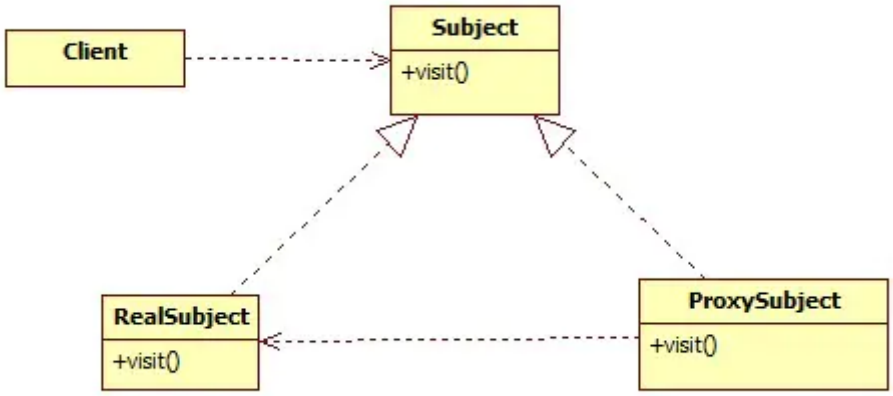
JDK **cglib**

3.1

3.1.1

1.
2.

3.1.2



1. Subject
2. RealSubject **Subject**
   Client
3. ProxySubject **Subject**

4. Client

> 1. Subject =
> 2. ProxySubject = = = =
> 3. RealSubject = = = =

3.1.3

> 1.
> 2.
> 3.

1.

```
1. public interface IRoom {
2.     void seekRoom();    //
3.     void watchRoom();   //
4.     void room();    //
5.     void finish();  //
6. }
```

2.

```
public class XiaoMing implements IRoom {
    @Override
    public void seekRoom() {
        System.out.println("      ");
    }

    @Override
    public void watchRoom() {
        System.out.println("      ");
    }

    @Override
    public void room() {
        System.out.println("            ");
    }

    @Override
    public void finish() {
        System.out.println("            ");
    }
}
```

3. IRoom

```
public class RoomAgency implements IRoom {

    private IRoom mRoom;    //

    public RoomAgency(final IRoom mRoom) {
        this.mRoom = mRoom;
    }

    @Override
    public void seekRoom() {
        mRoom.seekRoom();
    }

    @Override
    public void watchRoom() {
        mRoom.watchRoom();
    }

```

```
    @Override
    public void room() {
        mRoom.room();
    }

    @Override
    public void finish() {
        mRoom.finish();
    }
}
```

4.

```
public class Client {
    public static void main(String[] args) {
        //
        XiaoMing xiaoMing = new XiaoMing();
        //
        RoomAgency roomAgency = new RoomAgency(xiaoMing);
        //
        roomAgency.seekRoom();
        //
        roomAgency.watchRoom();
        //
        roomAgency.room();
        //
        roomAgency.finish();
    }
}
```

5.

XiaoHong XiaoHong IRoom `Client` `RoomAgency`

3.1.4

3.1.4.1

1.

3.2

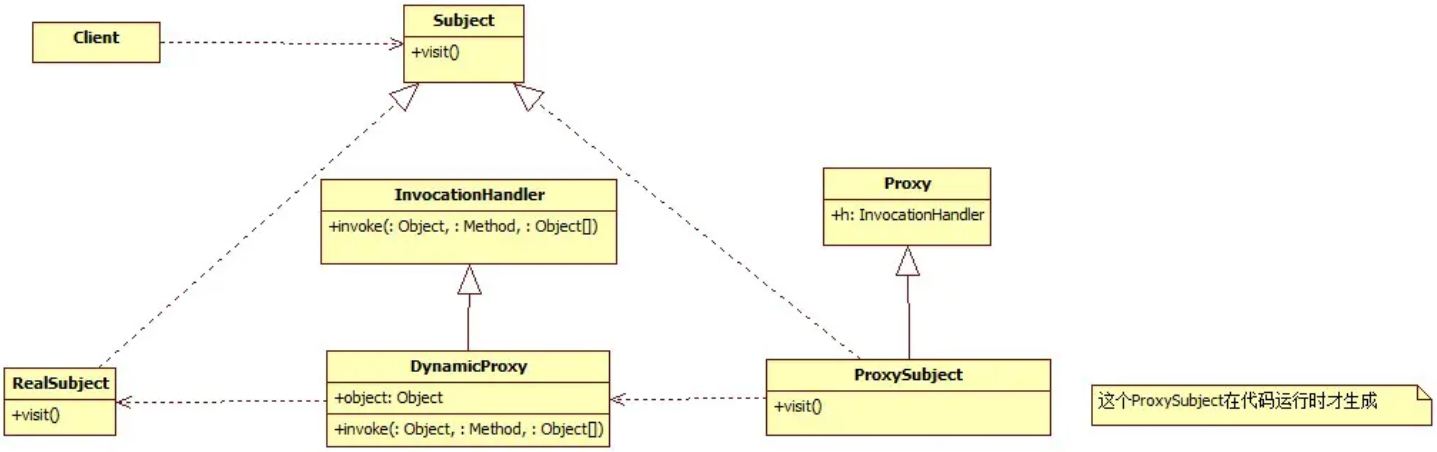
JDK cglib

3.2.1 JDK

3.2.1.1

1. JDK **JDK API** **class**

3.2.1.2



1. Subject ProxySubject RealSubject Client
2. InvocationHandler Proxy
   1. InvocationHandler Java **InvocationHandler** DynamicProxy
   2. Proxy Java **ProxySubject** **ProxySubject**
3. DynamicProxy ProxySubject RealSubject ProxySubject DynamicProxy DynamicProxy **RealSubject** ProxySubject

3.2.1.3

> JDK
>
> 1.
> 2.
> 3. **InvocationHandler**
> 4. **Proxy** **newProxyInstance**
>
> 1.
>   1. Proxy
> 2.
>   1. **InvocationHandler**
>   2. **Proxy** **newProxyInstance**

1.

```
// InvocationHandler
public class DynamicProxy implements InvocationHandler {

    private Object mObject; //

    public DynamicProxy(final Object mObject) {
        this.mObject = mObject;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        //
        Object result = method.invoke(mObject, args);
        return result;
    }
}
```

Object **invoke()** Object XiaoMing XiaoHong DynamicProxy XiaoHong DynamicProxy DynamicProxy

2. Proxy newProxyInstance()

```
public class Client {
    public static void main(String[] args) {
        //
        XiaoMing xiaoMing = new XiaoMing();
        //
        DynamicProxy dynamicProxy = new DynamicProxy(xiaoMing);
        //                    ClassLoader
        ClassLoader classLoader = xiaoMing.getClass().getClassLoader();

        // 1.      Proxy      newProxyInstance
        IRoom roomAgency = (IRoom) Proxy.newProxyInstance(classLoader, new Class[]{IRoom.class},
    dynamicProxy);

        // 2.
        //
        roomAgency.seekRoom();
        //
        roomAgency.watchRoom();
        //
```

```
        roomAgency.room();
        //
        roomAgency.finish();
    }
}
```

Proxy **newProxyInstance()** **Proxy**

1. **ClassLoader**
2. **IRoom** **IRoom** **Class**
3. **InvocationHandler** Proxy
   InvocationHandler
   InvocationHandler **invoke()** invoke()

3.2.1.4

Client 1

```
// 1.  Proxy  newProxyInstance
IRoom roomAgency = (IRoom) Proxy.newProxyInstance(classLoader, new Class[]{IRoom.class}, dynamicProxy);
```

1. Proxy newProxyInstance() Proxy newProxyInstance()

```
//Proxy.java
private static final Class<?>[] constructorParams = { InvocationHandler.class };

public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h) throws
IllegalArgumentException{
    Objects.requireNonNull(h);

    //clone
    final Class<?>[] intfs = interfaces.clone();

    //getProxyClass  getProxyClass0  getProxyClass  = getProxyClass0
        public  private
    //1  getProxyClass0  Class
    Class<?> cl = getProxyClass0(loader, intfs);

    try {

        //constructorParams = InvocationHandler.class
        //2  Class  InvocationHandler  Constructor
        final Constructor<?> cons = cl.getConstructor(constructorParams);
```

```
19.
20.     //         InvocationHandler
21.     final InvocationHandler ih = h;
22.     //       Constructor     protected                    Public
23.     if (!Modifier.isPublic(cl.getModifiers())) {
24.         cons.setAccessible(true);
25.     }
26.
27.     //3                   InvocationHandler     Constructor                                InvocationHandler
28.     return cons.newInstance(new Object[]{h});
29.   }
30.   //...
31. }
```

1. 1 getProxyClass0() Class getProxyClass0 getProxyClass .class Class ClassLoader Class Class Class Proxy Interface

```
1. public static Class<?> getProxyClass(ClassLoader loader, Class<?>... interfaces)throws
   IllegalArgumentException {
2.     final Class<?>[] intfs = interfaces.clone();
3.     final SecurityManager sm = System.getSecurityManager();
4.     if (sm != null) {
5.         checkProxyAccess(Reflection.getCallerClass(), loader, intfs);
6.     }
7.
8.     return getProxyClass0(loader, intfs);
9. }
```

2. Class
   1. Client main

```
1. // jdk8
2. System.setProperty("sun.misc.ProxyGenerator.saveGeneratedFiles", "true");
3. // jdk8
4. System.setProperty("jdk.proxy.ProxyGenerator.saveGeneratedFiles", "true");
```

   2. Client main idea com/sun/proxy/ $Proxy0.class Class .class JVM idea .java

```
1. public final class $Proxy0 extends Proxy implements IRoom{
2.   private static Method m1;
3.   private static Method m3;
```

```
4.  private static Method m4;
5.  private static Method m2;
6.  private static Method m5;
7.  private static Method m6;
8.  private static Method m0;
9.
10. //            Proxy                    InvocationHandler
11. public $Proxy0(InvocationHandler paramInvocationHandler){
12.     super(paramInvocationHandler);
13.  }
14.
15. //                          IRoom                                          h     invoke
    $Proxy0                        method
16. public final void watchRoom(){
17.   try{
18.     this.h.invoke(this, m3, null);
19.     return;
20.   }
21.   //...
22.  }
23.
24. public final void room(){
25.   try{
26.     this.h.invoke(this, m4, null);
27.     return;
28.   }
29.   //...
30.  }
31.
32. public final void seekRoom(){
33.   try{
34.     this.h.invoke(this, m5, null);
35.     return;
36.   }
37.   //...
38.  }
39.
40.
41. public final void finish(){
42.   try{
43.     this.h.invoke(this, m6, null);
44.     return;
45.   }
```

```
    //...
}

//...          IRoom                    Object          toSting  hashcode
                h    invoke

static{
    try{
        m0 = Class.forName("java.lang.Object").getMethod("hashCode", new Class[0]);
        m1 = Class.forName("java.lang.Object").getMethod("equals", new Class[] {
Class.forName("java.lang.Object") });
        m2 = Class.forName("java.lang.Object").getMethod("toString", new Class[0]);
        //      IRoom            Method
        m3 = Class.forName("com.example.hy.designpatternDemo.proxy.IRoom").getMethod("watchRoom",
new Class[0]);
        m4 = Class.forName("com.example.hy.designpatternDemo.proxy.IRoom").getMethod("room", new
Class[0]);
        m5 = Class.forName("com.example.hy.designpatternDemo.proxy.IRoom").getMethod("seekRoom",
new Class[0]);
        m6 = Class.forName("com.example.hy.designpatternDemo.proxy.IRoom").getMethod("finish", new
Class[0]);
        return;
    }
    //...
}
}
```

3. Proxy getProxyClass0() $Proxy0.class Proxy
   IRoom $Proxy0 IRoom
   h invoke() h InvocationHandler $Proxy0
4. $Proxy0 Method **$Proxy0** **static** **IRoom**
   Method **method** **$Proxy0**
   h **invoke()** **invoke()** **method**

```
// InvocationHandler
public class DynamicProxy implements InvocationHandler {

    private Object mObject; //

    public DynamicProxy(final Object mObject) {
        this.mObject = mObject;
    }

    @Override
```

```
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        //
        Object result = method.invoke(mObject, args);
        return result;
    }
}
```

2. Client 1 **Proxy** **newProxyInstance()** **$Proxy0** **$Proxy0** **InvocationHandler** **invoke()** InvocationHandler invoke() InvocationHandler

3.2.1.5

1. **JVM** **.class** **class**
2. **.class** **JVM** **class**
3. class

3.2.1.6

3.2.1.6.1

1.
2. InvocationHandler

3.2.1.6.2

1.
2.

1.
2. ——
3. ——
4.
5.
6.

2.1

1

1.

> log text

2.

>

3.

   1.

      1. + =

      2.

   2.

      1. **FileIO** CachedFileIO
      CachedIO new CachedIO(new FileIO())

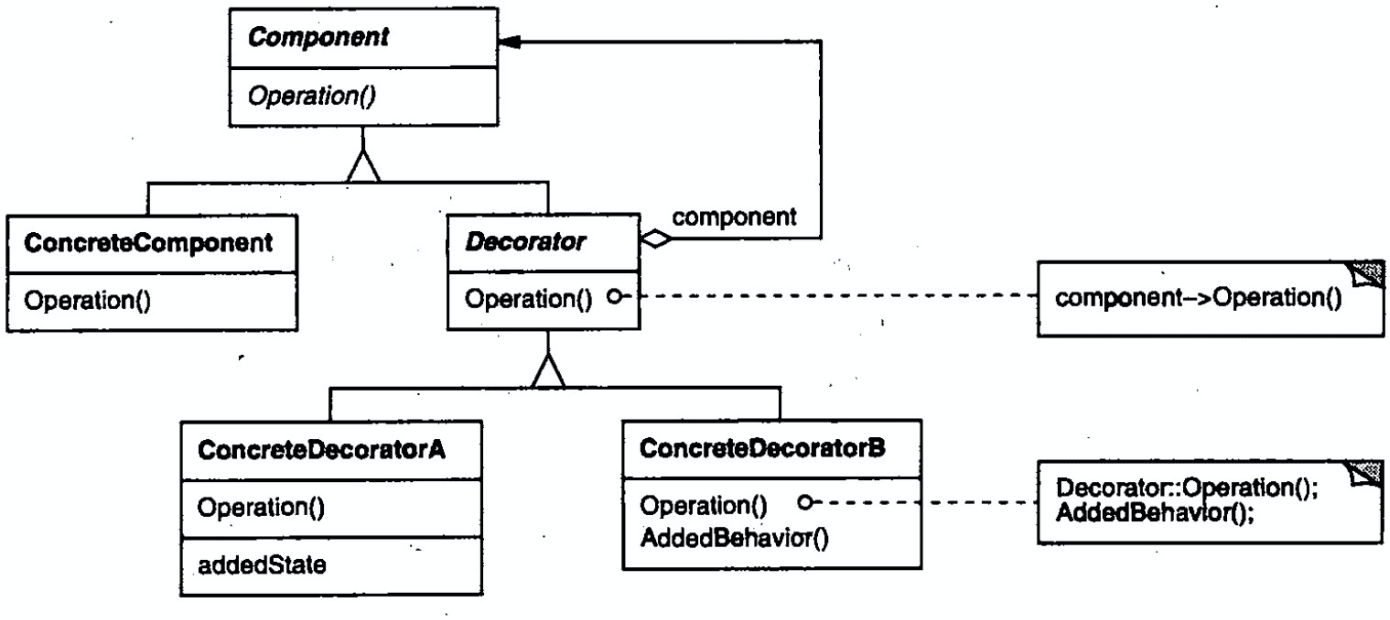
      2. NetworkIO DBIO
      new CachedIO(new NetworkIO())

      3. CachedIO **FileIO** **CachedFileIO** **CachedIO**

   3. ** wrapper **

      1. wrapper wrapper
      wrapper wrapper **API**

2



1. Component
2. ConrecteComponent
3. Decorator
4. ConcreteDecoratorA ConcreteDecoratorB

3

1.
    1.
    2.
2. Cook

```
public interface Cook {
    /**
     *
     */
    public void cookDinner();
}
```

3. ** ChineseCook **

```
public class ChineseCook implements Cook{
    /**
     *
     */
    @Override
```

```
    public void cookDinner() {
        System.out.println("                ");
    }
}
```

4. `FilterCook` `Cook` `Cook`

```
public abstract class FilterCook implements Cook{
    /**
     *
     */
    protected Cook cook;
}
```

5. `WashHandsCook` `WashHearCook`

```
public class WashHandsCook extends FilterCook{
    public WashHandsCook(Cook cook) {
        this.cook = cook;
    }

    /**
     *
     */
    @Override
    public void cookDinner() {
        washHand();
        this.cook.cookDinner();
    }

    /**
     *
     */
    private void washHand() {
        System.out.println("      ");
    }
}
```

```
public class WashHearCook extends FilterCook{
    public WashHearCook(Cook cook) {
        this.cook = cook;
    }
```

```
5.
6.    /**
7.     *
8.     */
9.    @Override
10.   public void cookDinner() {
11.       washHear();
12.       this.cook.cookDinner();
13.   }
14.
15.   /**
16.    *
17.    */
18.   private void washHear() {
19.       System.out.println("      ");
20.   }
21. }
```

6. Client

```
1. public class Client {
2.     public static void main(String[] args) {
3.         Cook cook1 = new WashHandsCook(new ChineseCook());
4.         Cook cook2 = new WashHearCook(new ChineseCook());
5.
6.         cook1.cookDinner();
7.         cook2.cookDinner();
8.     }
9. }
```

4

4.1

1.

2.

4.2

1.

    1.

2. **bug**

5

1.

2.
3.

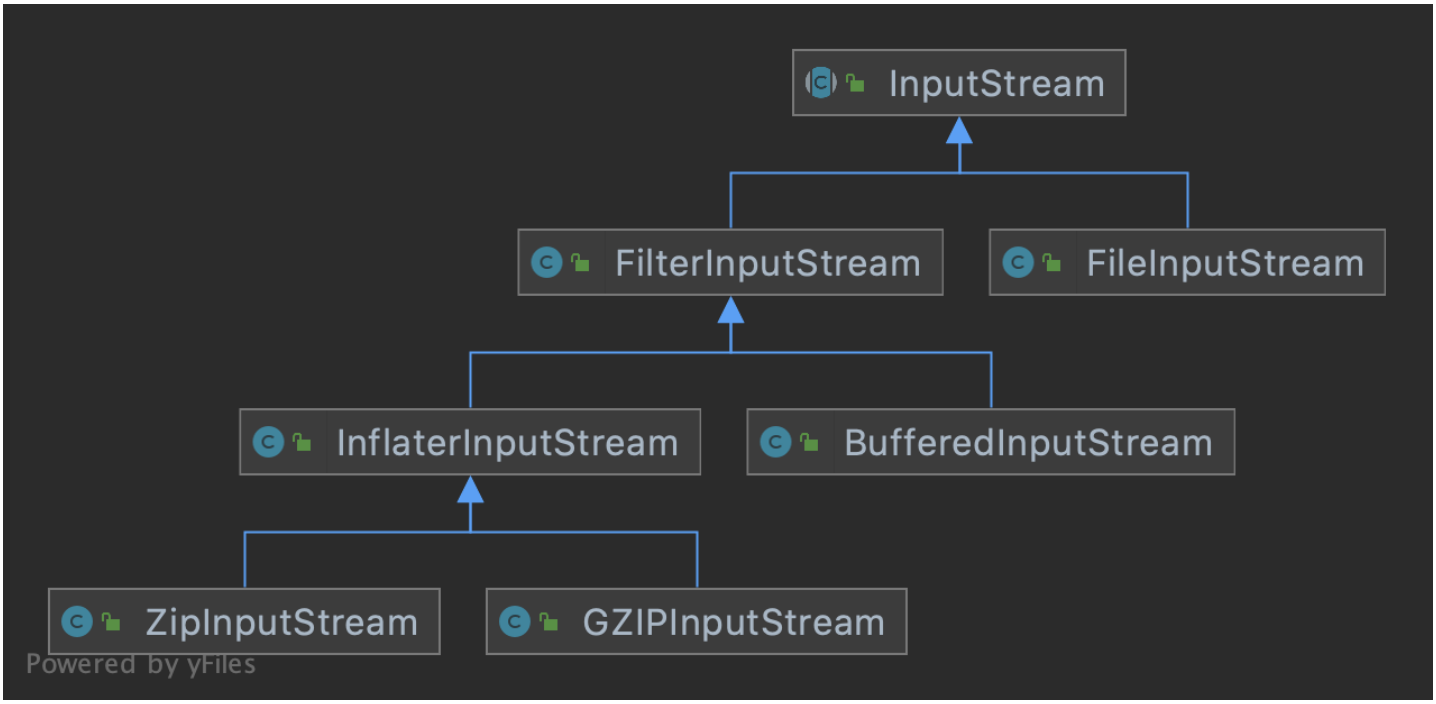
6

6.1 JDK

1. **Java I/O** java.io **InputStream/OutputStream** **Reader/Writer**

   1. InputStream FilterInputStream
   2. OutputStream FilterOutputStream
   3. Reader BufferedReader FilterReader
   4. Writer BufferedWriter FilterWriter PrintWriter

2. InputStream InputStream FileInputStream ObjectInputStream ByteArrayInputStream



1. InputStream
2. FileInputStream ObjectInputStream
3. FilterInputStream **InputTsream**

4. `FilterInputStream` `BufferedInputStream`

---

1. Java 12
2. Decorator
3. Java —-
4. -
5.
6. ( ) (Decorator)

3.0

1

1.
2.
3.

   `if...else...`

4.
5.

   Strategy

2

1. Strategy Pattern
2. Policy Pattern

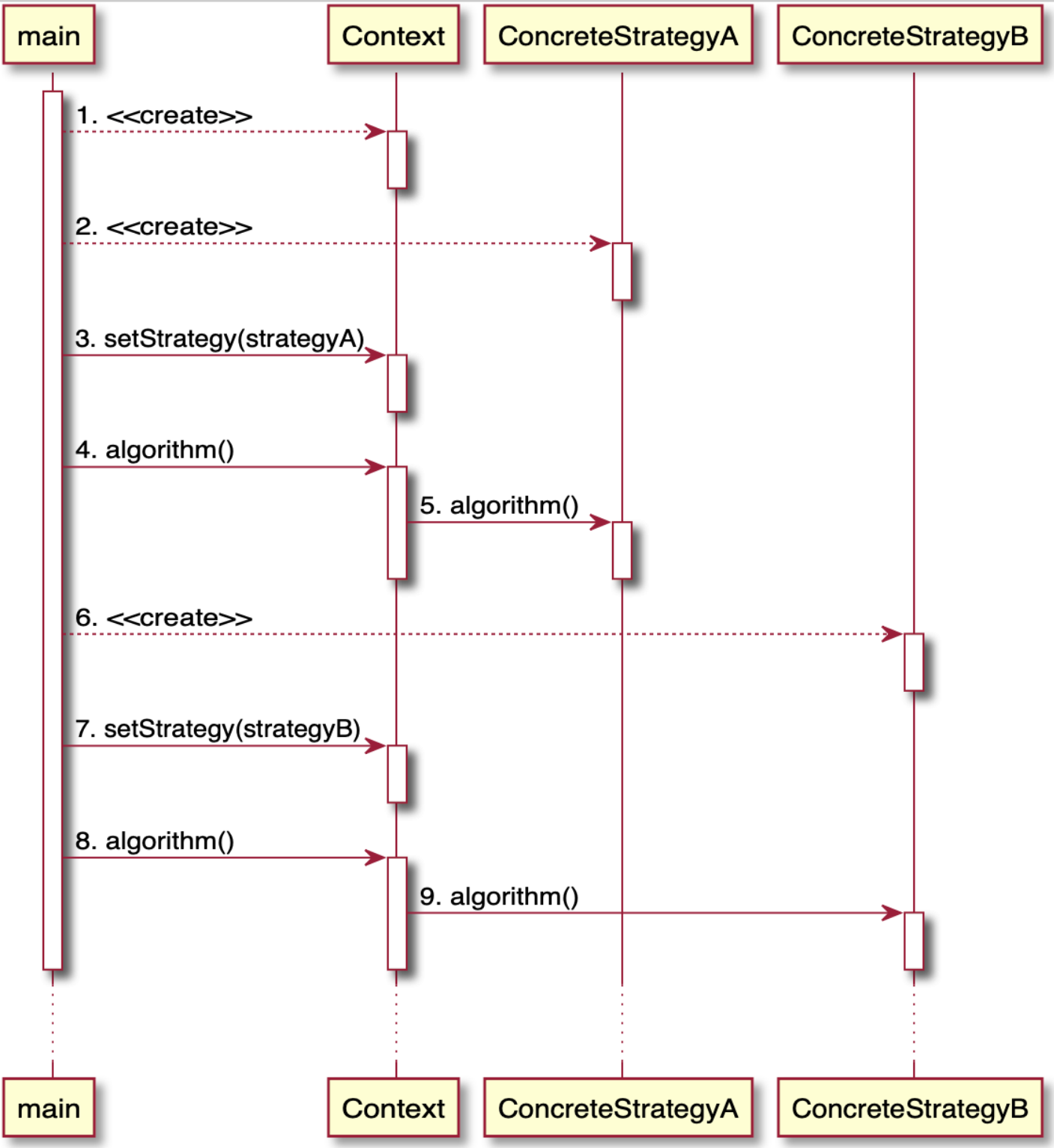
3

1. Strategy
2. ConcreteStrategy
3. Context ../_images/Strategy.jpg

4

1.
2.
3.

4.


main
Context
ConcreteStrategyA
ConcreteStrategyB
1. <<create>>
2. <<create>>
3. setStrategy(strategyA)
4. algorithm()
5. algorithm()
6. <<create>>
7. setStrategy(strategyB)
8. algorithm()
9. algorithm()
main
Context
ConcreteStrategyA
ConcreteStrategyB

5

5.1

CashStrategy acceptCash()

```
abstract class CashStrategy {
    /**
     *
     * @param money
     * @return
     */
    public abstract double acceptCash(double money);
}
```

5.2

CashNormalStrategy CashReturnStrategy CashRebateStrategy CashStrategy acceptCash()

```
public class CashNormalStrategy extends CashStrategy{
    /**
     *
     * @param money
     * @return
     */
    @Override
    public double acceptCash(double money) {
        System.out.println(String.format("                %s.", money));
        return money;
    }
}
```

```
public class CashReturnStrategy extends CashStrategy{
    private double moneyCondition = 0.0;
    private double moneyReturn = 0.0;

    public CashReturnStrategy(final double moneyCondition, final double moneyReturn) {
        this.moneyCondition = moneyCondition;
        this.moneyReturn = moneyReturn;
    }

    /**
     *
     * @param money
     * @return
```

```
     */
    @Override
    public double acceptCash(double money) {
        double res = money;
        if (money >= moneyCondition) {
            res = money - Math.floor(money / moneyCondition) * moneyReturn;
        }
        System.out.println(String.format("    %s     %s            %s.", moneyCondition, moneyReturn, money));
        return res;
    }
}
```

```
public class CashRebateStrategy extends CashStrategy{
    private double moneyRebate = 1.0;

    public CashRebateStrategy(final double moneyRebate) {
        this.moneyRebate = moneyRebate;
    }

    /**
     *
     * @param money
     * @return
     */
    @Override
    public double acceptCash(double money) {
        double res = money * moneyRebate;
        System.out.println(String.format("                    %s          %s.", moneyRebate, money));
        return res;
    }
}
```

5.3

```
public class CashContext {
    private CashStrategy cashStrategy;

    public CashContext(String type) {
        switch (type) {
            case "0": cashStrategy = new CashNormalStrategy();break;
```

```
            case "           ": cashStrategy = new CashReturnStrategy(300.0, 100.0); break;
            case "8    ": cashStrategy = new CashRebateStrategy(0.8); break;
        }
    }

    /**
     *
     * @param money
     * @return
     */
    public double getResult(double money) {
        return cashStrategy.acceptCash(money);
    }
}
```

5.4

```
public class Client {
    public static void main(String[] args) throws InterruptedException {
        Scanner scanner = new Scanner(System.in);
        String type = null;
        double money = 0.0, res = 0.0;

        System.out.println("Please input the discount(0/           /8    /      ): ");
        type = scanner.nextLine();
        while (!type.equals("      ")) {
            CashContext cashContext = new CashContext(type);
            System.out.println("Please input money: ");
            money = scanner.nextDouble();
            res = cashContext.getResult(money);
            System.out.println(String.format("                    %s", res));
            System.out.println("Please input the discount(0/           /8    /      ): ");
            type = scanner.nextLine();
            type = scanner.nextLine();
        }
    }
}
```

6

6.1

1.
2.
3.
4.

6.2

1.
2.

7

1.
2.
3.
4.

1.
2. 5.
3.
4. 2.