Beijing-Dublin International College

SEMESTER 1 FINAL EXAMINATION - 2019/2020

COMP3007J Design Patterns

MODULE COORDINATOR: Junbiao Pang

Time Allowed: 95 minutes

Instructions for Candidates

BJUT Student ID: _____UCD Student ID: _____

I have read and clearly understand the Examination Rules of both Beijing University of Technology and University College Dublin. I am aware of the Punishment for Violating the Rules of Beijing University of Technology and/or University College Dublin. I hereby promise to abide by the relevant rules and regulations by not giving or receiving any help during the exam. If caught violating the rules, I accept the punishment thereof.

Honesty Pledge：_____ (Signature)

Instructions for Invigilators

All electronic devices, notebooks, books, work papers are strictly prohibited.

1. Directions: choose the best answer for each numbered question and mark A, B, C, or D on **ANSWER SHEET.** （40 points）

1. Which the statement about the Single Responsibility Principle(SRP) is WRONG ( ):

A. A class is only responsible for one responsibility in a functional domain.

B. It should be only one reason to change a class.

C. The more responsibilities a class has, the easier it is to be reused, and the more likely it is to be reused.

D. When a class has many responsibilities, it needs to separate the responsibilities and further encapsulate the different responsibilities in different classes.

2. To achieve object-oriented analysis and design, (1) is a module that should be open in terms of scalability but yet be closed in terms of maintainability; meanwhile, (2) means that subclasses could be able to replace the parent class and further could replace the parent class in a program.

(1). A. Open Closed Principle B. Liskov Substitution Principle

C. Dependence Inversion Principle D. Single Responsibility Principle

(2) A. Open Closed Principle B. Liskov Substitution Principle

C. Dependence Inversion Principle D. Single Responsibility Principle

3. Which the following statement is WRONG ( ):

A. The high-level modules should not depend on the low-level ones.

B. Abstraction should not depend on the details.

C. The details could depend on abstraction.

D. The high-level modules have to depend on the low-level ones.

4. For the Interface Segregation Principle, which the following statement is WRONG ( ):

A. Clients should not depend on the unwanted interfaces when we design a software.

B. When an interface has too many methods, it is better to reconstruct this interface into several ones which have a few methods; the advantage is that clients just needs to know the relevant methods.

C. Ideally, each interface should only define one function; by this way, the resulting interface would be easily to be used.

D. An interface represents only one role, and each role has a specific interface.

5. A software should interact with other software as few as possible. Because when a module is modified, it will affect other modules as little as possible; besides, it would make the extendibility of software easy. This is the definition of ( ):

A. Law of Demeter Principle B. Interface Segregation Principle

C. Liskov Substitution Principle D. Aggregation Reuse Principle

6. Which the following statement of the simple factory method pattern is WRONG ( ):

A. Simple factory method pattern could return instances of the different classes depending on the inputted parameters.

B. Simple factory method pattern specifically defines a class to be responsible for creating instances of other classes, and the created instances usually have a common parent class.

C. Simple factory method pattern can reduce the number of classes in a software and thus simplify the design of the software.

D. The expendability of a software is limited. If you add a new product, you have to modify the logic of the factory, which would violate the Open and Closed Principle.

7. The following code uses the ( ) pattern:

A. Simple Factory

B. Factory Method

C. Abstract Factory

D. No design patterns are used

```
public abstract class ExchangeMethod {
   public abstract void process ( );
}

public class DigitalCurrency extends ExchangeMethod{
 public void process( ){...}
}

public class CreditCard extends ExchangeMethod{
   public void process( ){...}
}
...

public class Factory{
   public static ExchangeMethod createProduct(String type){
     switch(type){
       case"DigitalCurrency":
       return new DigitalCurrency( );break;
       case"CreditCard":
       return new CreditCard( );break;
```

```
        ...
    }
    }
}
```

8. Fig. 1 is the class diagram of the ( ) pattern.

A. Abstract Factory B. Factory Method

C. Command D. Chain of Responsibility

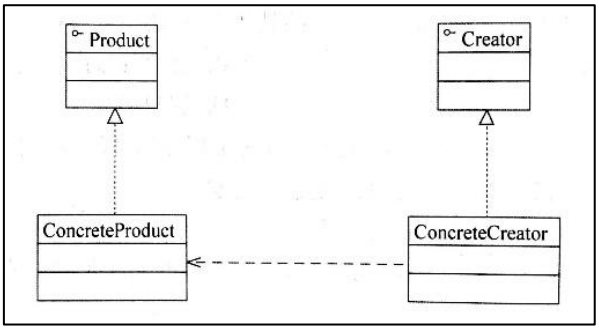

Fig.1

9. A bank system uses the factory method pattern to model the relationship between different accounts. The class diagram is shown in Fig.2. The class corresponding to the "creator" in the factory method pattern is (1); the class corresponding to the "product" is (2).

(1) A. bank B. Account C. Checking D. Savings

(2) A. bank B. Account C. Checking D. Savings

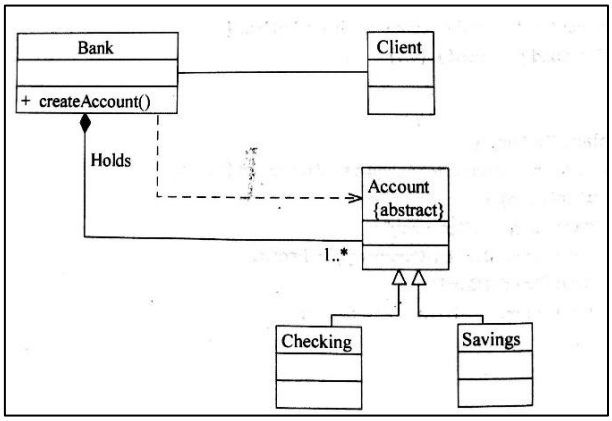

Fig. 2

10. When a product has a complex internal structure, in order to gradually construct this object and further make it more extendible, you could use ( ):

A. Abstract Factory Method Pattern B. Prototype Pattern

C. Builder pattern D. Singleton Pattern

11. A company plans to develop a live chat software that allows users to communicate with multiple friends at the same time in a public window, and generates a new chat window for a

friend if the private chat is involved. To increase efficiency of this system, when a friend requires a private chat, you need to quickly create a new chart window based on the public window. For this demand, ( ) would be used.

A. Flyweight pattern B. Singleton Pattern

C. Prototype Pattern D. Composite Pattern

12. The ( ) pattern combines multiple objects into a tree structure to represent a "part-whole" hierarchy, and also makes users uniformly treat the individual objects and the composite objects.

A. Composite B. Bridge C. Decorator D. Facade

13. (1) pattern separates the abstract part from its implementation, so that these classes could change independently. Fig.3 shows the class diagram for this pattern, where (2) is used to define the interface.

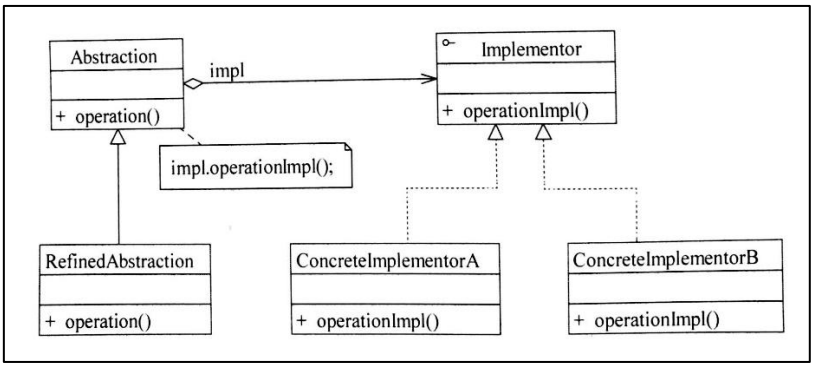

Fig. 3

（1） A. Singleton B. Bridge C. Composite D. Facade

（2） A. Abstraction B. ConcreteImplementorA

C. ConcreteImplementorA D. Implementor

14. Now, we need to develop an XML processing software, which can query the specified content according to the supplied keyword. The user can select a certain node in the XML as the initial node for a query; meanwhile, users do not need to care about the hierarchical structure of this node. For this requirement, we can use ( ) pattern.

A. Abstract Factory B. Flyweight C. Composite D. Strategy

15. Which of the following statement is NOT reasonable in Fig. 4 ( )

A. Dynamically determine which object from a set of ones to handle a request.

B. Dynamically assign a request to a set of objects, and the request would be efficiently handled.

C. make multiple objects have the opportunity to handle a request, and decouple the sender and the receiver.

D. organize objects into a chain, and pass a request along this chain.

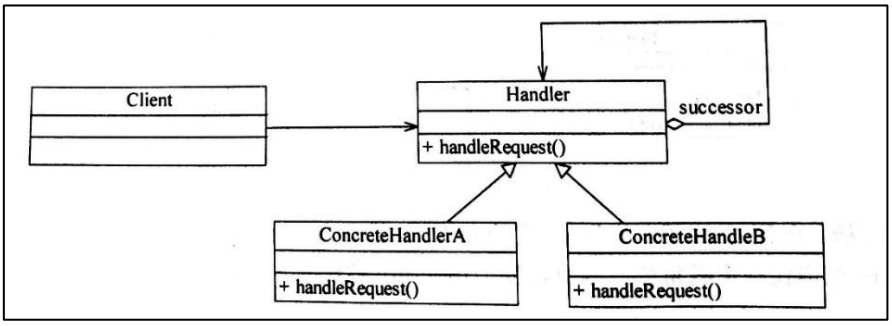

Fig.4

16. Fig. 5 shows the class diagram of the calculator which uses command pattern. (1) acts as the request caller, and (2) acts as the charge request receiver.

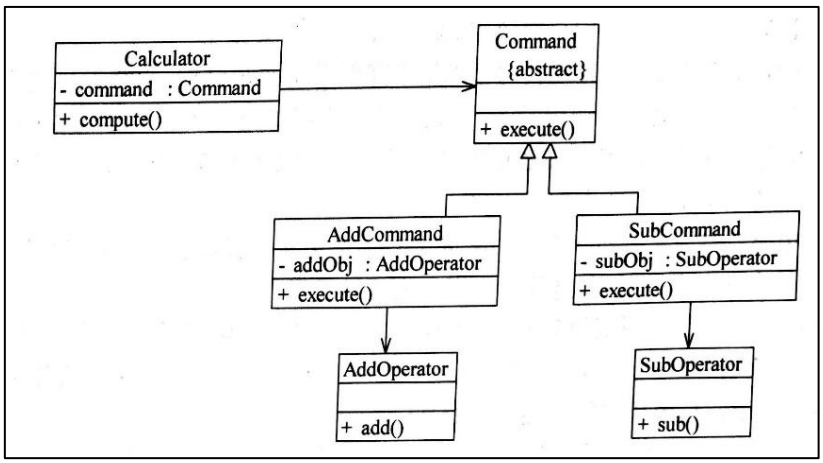

Fig.5

(1) A. Calculator B. Command C. AddCommand D. AddOperator

(2) A. Calculator B. Command C. AddCommand D. AddOperator

2. Directions: Read the following text, some code segments have been removed. Please complete these programs to make sure them output the correct results. **More importantly, all programs should FOLLOW the principle of the object-oriented programming principles.** Write the answer for each numbered blank on **ANSWER SHEET.** （40 points）

1、A company wants to develop a data format transfer tool that can convert a data between different data sources, such as, change TXT files, databases, and Excel tables, into XML format.

In order to make the system more scalable, we wish that new data sources could be supported in the future. Therefore, developers intend to use the Factory Method Pattern to design the core class of the conversion tool. Factory class encapsulates the initialization and creation process for some data types, as shown in Fig. 6.

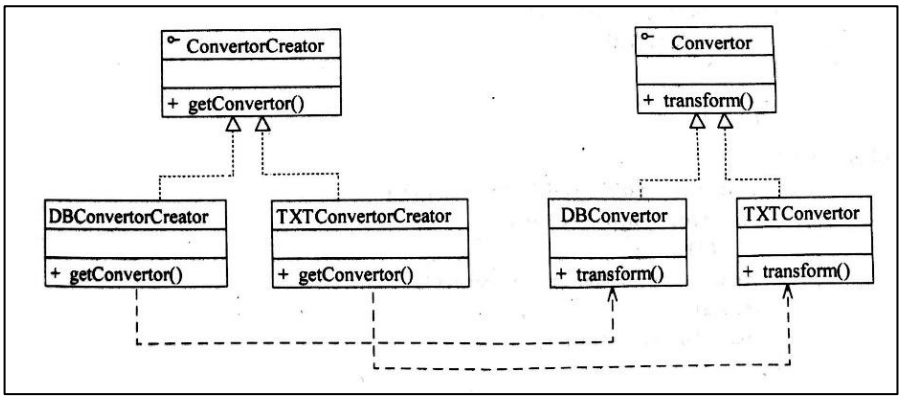

Fig.6

In the figure, ConvertorCreator is the interface of an abstract factory that declares the factory method getConvertor( ), which is implemented in its subclasses; Convertor is the interface of the abstract product which declares the abstract data transformation method transform ( ). The classes DBConvertor and TXTConvertor are individually used to convert data stored both in the database and in the TXT file to the XML format.

```
interface ConvertorCreator{
                ( 1 );
}

interface Convertor{
                public String transform ( );
}

Class DBConvertorCreator implements ConvertorCreator
                public Convertor getConvertor( ){
                ( 2 );
                }
}

class TXTConvertorCreator implements ConvertorCreator{
                public Convertor getConvertor( ){
                ( 3 );
                }
}

class DBConvertor implements Convertor{
                public String transform( ){
                //Implementation code omitting
                }
```

```
}

class TXTConvertor implements Convertor{
        public String transform( ){
            //Implementation code omitting
        }
}

class Test{
        public static void main(String args[ ]){
            ConvertorCreator creator;
            ( 4 );
            creator=new DBConvertorCreator( );
            convertor=( 5 );
        convertor.transform( );
      }
}
```

If you need to convert data for a new type of data source, the system needs to add at least **(6)** classes. Which the object-oriented design principles are used in the factory method pattern? **(7)** (multiple choices).

A. Open and Closed Principle

B. Dependence Inversion Principle

C. Interface Segregation Principle

D. Single Responsibility Principle

E. Composite Reuse Principle

2. To design an image browsing system, which is expected to display an image with the BMP, JPEG or GIF formats on both Windows and Linux operating systems. The system firstly parses the images with the BMP, JPEG, or GIF formats into the pixel matric, and then displays the pixel matric on the displayer. The system is required to be well scalable for new file formats and operating systems.

In order to meet these requirements and reduce the number of subclasses, the resulting class diagram is shown in Fig.7, where (1) pattern is used. The reason for adopting this design pattern is that the classes for parsing BMP, JPEG, and GIF files are only related to the file format. But, the code for displaying pixels on the displayer is only related to the operating systems.

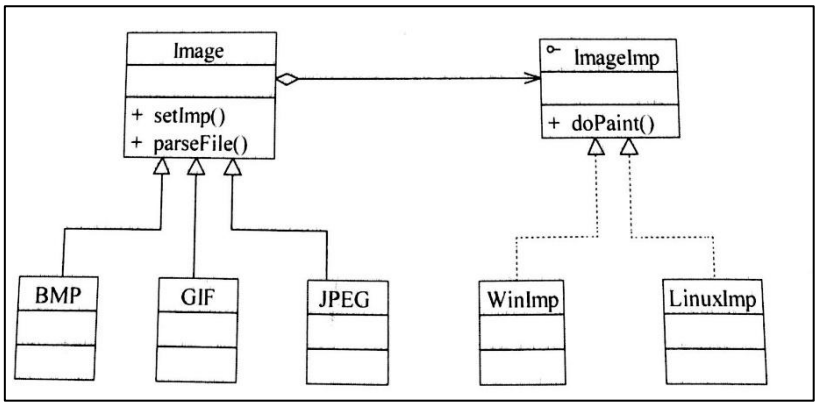

Fig.7

```
class Matrix{//Files in various formats are eventually converted into pixel matric

        //Code is omitted here
}

interface ImageImp{
        public void doPaint(Matrixm); //Display pixel matrixm
}

class WinImp implements ImageImp{
public void doPaint(Matrixm){/* Call the drawing function of the
                                        Windows system to draw the pixel matrix */)
}

class LinuxImp implements ImageImp{
        public void doPaint(Matrix m){/* Call the drawing function of
                                        the Linux system to draw the pixel matrix */}
}

abstract class Image{
        public void setImp(ImageImp imp){
        ( 2 )=imp;}
        public abstract void parseFile(String fileName);
        protected( 3 )imp;
}

class BMP extends Image{
        public void parseFile(String fileName){
                        //Parse the BMP file here and get a pixel matrix object m
        ( 4 )           //Display pixel matrix m
        }
}

class GIF extends Image{
        //Code omitted here
}
```

```
class JPEG extends Image{
        //Code omitted here
}

public class Main{
        public static void main(String[ ]args)
        {
                //View the demo.bmp image file on the Windows operating system
                Image imagel=( 5 );
                ImageImp imageImp1=( 6 );
                ( 7 );
                imagel.parseFile("demo.bmp");
        }
}
```

Now suppose that the software needs to support 10 types of image files and 5 types of the operating systems, without considering Matrix class and Main class. If we still use the pattern in Fig. 7, the number of classes at least should be (8).

3.The procurement of an enterprise is approved by the different level of managers. That is, different levels of supervisors would be granted to approve the orders with the different amounts.

More specially, the director can only approve the amount of an order which is less than 50,000 $ (excluding 50,000$), the vice chairman can approve the amount of an order which is between 50,000$ and 100,000$ (excluding 100,000$), and the chairman can approve the amount of an order which is between 100,000$ and 500,000$ (excluding 500,000$). An order with more than 500,000$ should be discussed by all people.

By the chain of responsibility pattern, the following class diagram is illustrated in Fig. 8.

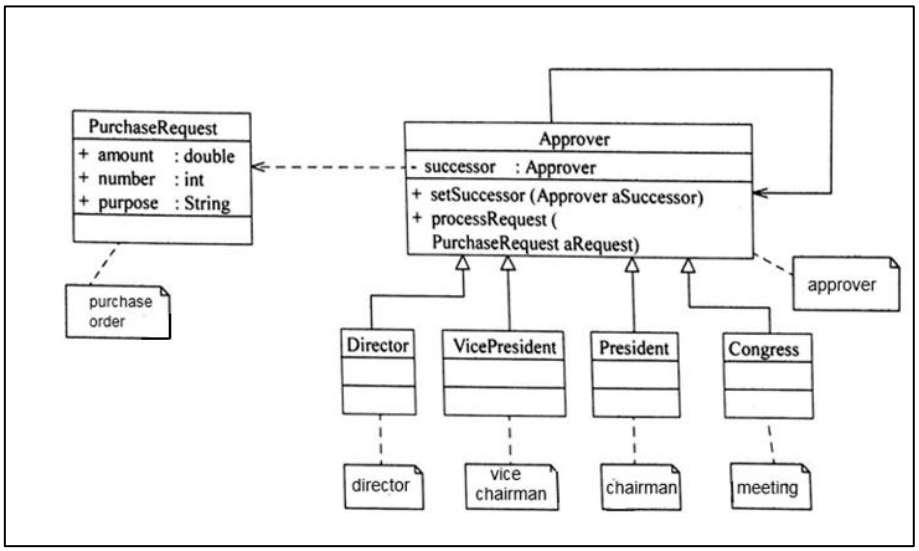

Fig.8

```
class PurchaseRequest{
        public double amount;
        public int number;
        public String purpose;

//Approver class
class Approver{
        public Approver( ){ successor=null;}
        public void processRequest(PurchaseRequest aRequest){
                if(successor!=null){successor.( 1 );}
        public void setSuccessor(Approver aSuccessor){successor=aSuccessor;)
        private( 2 )successor;
        }

class Congress extends Approver{
        public void processRequest(PurchaseRequest aRequest){
                if(aRequest.amount>=500000){/*Code omitted here*/}
                else ( 3 ).processRequest(aRequest);
        }
}

class Director extends Approver{
        public void processRequest(PurchaseRequest aRequest){/*Code omitted here*/}
}

class President extends Approver{
        Public void processRequest(PurchaseRegiest aReguest){/*Code omitted here*/}
}

class VicePresident extends Approver{
        Pubic void processRequest(PurchaseReguest aReguest)/*Code omitted here*/}
}

public class Test{
        public static void main(String[]args)throws IOException{
                Congress meeting=new Congress( );
                VicePresident sam=new VicePresident( );
                Director larry=new Director( );
                President tammy=new President( );
                meeting.setSuccessor(null);
                sam.setSuccessor(( 4 ));
                tammy.setSuccessor(( 5 ));
                larry.setSuccessor(( 6 ))
                //Construct a procurement approval request
                PurchaseRequest aRequest=new PurchaseRequest( );
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                aRequest.amount=Double.parseDouble(br.readLine( ));
                ( 7 ).processRequest(aRequest);
        } }
```

3. Directions: The following requirements is from the different clients. **By FOLLOWING the principle of the object-oriented programming principles**, you, as a software engineer, should: 1) give the solution in a UML class diagram; and 2) write the corresponding JAVA code on **ANSWER SHEET**. (20 points）

1. Haier、TCL and Hisense are all electrical appliance manufacturers. They produce Televisions, Air Conditioners, and Refrigerators. A software is needed to manage these electrical appliance manufacturers and the appliances they produce.

   It is required to draw a class diagram and implement it with JAVA.

2. A software company intends to develop a software to generate electronic music. This software is expected to use an algorithm to simulate one instrument and further generate its sounds. Naturally, this software needs multiple algorithms to generate the sounds of different instruments.

   Besides, the generated sounds should be exported into different audio format, for example, mp3, wma, and Cda etc.

   According to the above description, please select a suitable design pattern to design this software.