BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
FIRST SEMESTER, 2018-2019
COMPREHENSIVE EXAM (CLOSED BOOK)
PART I - OBJECT ORIENTED PROGRAMMING (CS F213)

DATE: 12th December 2018 **MAX MARKS: 22.5** **TIME: 60 Mins.**

Important Instructions:

1. The exam comprises of Part I, Part II and Part III. Part I and II are closed book and Part III is open book. The maximum duration for Part I is 1 hr. You can receive the Part III question paper once you have submitted the Part II.
2. **Attempt all questions. Each question carries 0.75 mark.**
3. **More than one option may be correct. Choose all the correct option(s) to get any credit.**
4. **For every incorrect answer 0.25 marks will be deducted.**
5. **Overwriting and cutting is not acceptable and will not be evaluated.**

Q.1. Encapsulation

Q.2. If you want to disallow instantiation of that class from outside that class

Q.3. The code does not compile

Q.4. void actionPerformed(ActionEvent e)

Q.5. NullPointerException

Q.6. The catch blocks for exceptions of type B and C are unreachable.

Q.7. this program does not compile

Q.8. 3,3,3

Q.9. float [][] someArray

Q.10. II and III

Q.11. I and II

Q.12. I, II, and III

Q.13. Static methods cannot be synchronized
Variables can be protected from concurrent access by using synchronized keyword.
When a thread sleeps, it releases the locks.

Q.14. II only

Q.15. 15 23 21 19 19 21 23 15

Q.16. 2

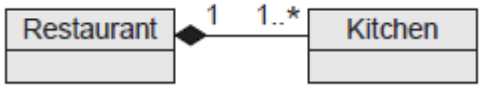
Q.17. wait() method is overloaded to accept a duration.
Both wait() and notify() must be called from a synchronized context.

Q.18. private synchronized int e;

Q.19. Fails to compile because the IllegalMonitorStateException of wait()

Q.20. 20

Q.21.



Q.22. this{statements;}

Q.23. null, One

Q.24. String s = Integer.toString(a)
String s = String.valueOf(a)

Q.25. 120, 80, 100, 80

Q.26. False

Q.27. No

Q.28. Box<String> b = new Box<String>();
Box<String> b = new SuperBox<String>();

Q.29. Compilation error in list3

Q.30. Inside B

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
FIRST SEMESTER, 2018-2019
COMPREHENSIVE EXAM (CLOSED BOOK)
PART II - OBJECT ORIENTED PROGRAMMING (CS F213) - SOLUTIONS

DATE: 12th December 2018 **MAX MARKS: 27.5** **TIME: 60 Min**

Q1. a)
```
count++;
return screen.get(pos);
```
[1M]

Q1.b)
```
name = Thread.currentThread().getName();
for(i = 0; i<10; i++)
      card.add((int)(Math.random() * 50));
```
[0.5M]
[0.5M]

Q1.c)
```
i<10 & msg.winner.equals("No one")
```
[1M]

Q1.d)
```
synchronized(msg) {
                while(msg.available==false | (i+1) > msg.screen.size())
                {
                      try {
                            msg.wait();
                      } catch (InterruptedException e) {}
                }
                token = msg.getMsg(i);
                if (msg.count==msg.noOfPlayers)
                {
                      msg.available =false;
                      msg.count =0;
                      msg.notifyAll();
                }

                 if(card.contains(token))
                 {
                      match++;
                      card.remove(card.indexOf(token));
                 }
                if(match==3)
                {
                      msg.won(name);
                }
          }
```
[0.5M]
[2M]
[0.5M]
[2M]
[2M]
[1M]

Q1.e)
```
name = Thread.currentThread().getName();
```
[0.5M]

Q1.f)
```
i<10 & msg.winner.equals("No one")
```
[1M]

Q1.g)
```
number = (int)(Math.random() * 50);
```
[0.5M]

Q1.h)
```
synchronized(msg) {
                while(msg.available==true & msg.count !=0 ){
                      try {
                            msg.wait();
                      } catch (InterruptedException e) {}}
                msg.setMsg(number);
                try {
                      Thread.sleep(1000);
                            } catch (InterruptedException e) {}

                msg.available=true;
                 msg.notifyAll();
                }
```
[0.5M]
[1.5M]
[0.5M]
[0.5M]
[0.5M]
[0.5M]

Q1.i) [1M]

```
moderator = new Thread(new Moderator(msg), "Moderator");
Player1 = new Thread(new Player(msg),"Player1");
Player2 = new Thread(new Player(msg), "Player2");
```

[1M]

```
Moderator.start();
Player1.start();
Player2.start();
```

[1M]

```
Player2.join();
Player1.join();
Moderator.join();
```

Q2.a) [1M]

```
if (username.contains("@gmail.com")) {
                this.username = username;
                success = true; }
        else  {
                success = false;
                throw new CheckFormatException("Not a gmail ID"); }
```

Q2.b) [1M]

```
if(!password.toLowerCase().equals(password))   {
             this.password = password;
             success = true;   }
         else  {
             success = false;
             throw new CheckFormatException("Use atleast one capital letter in password");
         }
```

Q2.c) [1M]

```
class CheckFormatException extends RuntimeException {

          CheckFormatException(String message) {
              Super(message); }
}
```

Q2.d) [2M]

```
name = fname+" "+lname;

         if(!dB1.containsKey(name))     {
                e.success = true;
                e.setUser(username);
                e.setPass(password);
                dB1.put(name, username);
                dB2.put(username, password);}
         else {
                e.success = false;
                throw e.new CheckFormatException("Data already exists");
         }
```

Q2.e)

```
in = new Scanner(System.in);
noOfPersons = in.nextInt();
Person P[] = new Person[noOfPersons];
```

[0.5M] [0.5M] [0.5M]

Q2.f)

```
P[i].createAccount(in.next(), in.next(), in.next(), in.next());
```

[1M]

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
FIRST SEMESTER, 2018-2019, COMPREHENSIVE EXAM (OPEN BOOK TEST)
PART-III, OBJECT ORIENTED PROGRAMMING (CS F213)

DATE: 12/12/2018 **MAX MARKS: 30** **TIME: 60 Min**

Instructions for Q.1 (a) to Q.1 (c) – Write the name of the software design pattern that you think can be appropriately applied in a given context and then write your justification for that in no more than 200 words. [IMPORTANT: You will lose marks for being verbose]

Q.1 (a) A server receives messages from a client containing operations that the server should do. The server is supposed to execute these operations in the order received. **What design pattern should be used in the server's implementation to accomplish this?**

ANSWER: **COMMAND**

Q.1 (b) You want to add the ability to write text to **Java standard output** [using `System.out.println()`] or a **file** in all UPPERCASE. You want this to work with client code that can write text to a PrintStream, but it writes the text in all LOWERCASE and requires conversion. **What pattern should you use and why?**

ANSWER: **DECORATOR**

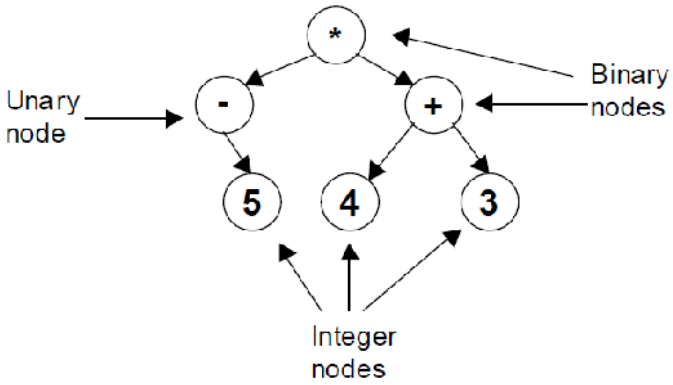
It is not Adapter because, in the *Adapter* pattern, you have the correct functionality in the original object, but *need* to *use* a different interface. In the *Decorator* pattern, you want to *use* the original interface, but also to add functionality.

Q.1 (c) Suppose you are implementing some application (say **XImpl**) for an android mobile phone and you have created an object to keep track of whether the phone is in airplane mode. There are many other applications that need to know when the phone goes into airplane mode so that they can start running in low-power mode to preserve your battery life and when it comes out start running normally, but you do not know exactly what those applications are. **What pattern should you use and why? Draw a UML class diagram by choosing appropriate names for the classes for XImpl application.**

ANSWER: **OBSERVER**

[2 + 2 + 6 = 10 Marks]

Q.2. Expression tree consist of nodes containing operators and operands. Operators have different precedence levels; e.g., multiplication takes precedence over addition. The multiplication operator has two arguments, whereas unary minus operator has only one. Operands are integers, doubles, variables, etc. We will deal with integers in this question. Expression tree may be "evaluated" via different traversals, e.g., in-order, post-order, pre-order, level-order. The evaluation step may perform various operations, e.g., traverse and print the expression tree, return the value of the expression tree. Consider the example of the expression tree given below:



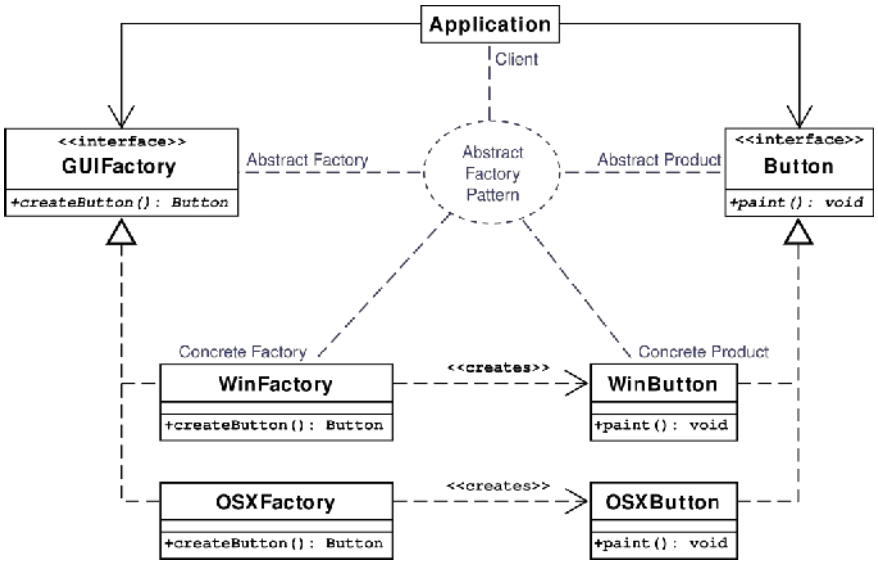
Show how Abstract Factory could be effectively used in this application. Firstly, show the UML class diagram. Use proper class names and some of the important operations and instance variables required for use. Secondly, show some code snippets in java for each class in your design and illustrate how the nodes would be created. **[20 Marks]**

ANSWER:

Explanation [optional – only for clarity]

The abstract factory pattern is a software design pattern that provides a way to encapsulate a group of individual factories that have a common theme. In normal usage, the client software creates a concrete implementation of the abstract factory and then uses the generic interfaces to create the concrete objects that are part of the theme (courtesy Wikipedia).

Essentially, the abstract factory pattern provides an interface for creating families of related or dependant objects, without exposing their concrete classes.



https://en.wikipedia.org/wiki/Abstract_factory_pattern

```
public static GUIFactory createOsSpecificFactory() {
    int sys = readFromConfigFile("OS_TYPE");
    if (sys == 0) {
        return new WinFactory();
   } else {
```

```
            return new OSXFactory();
        }
    }
```

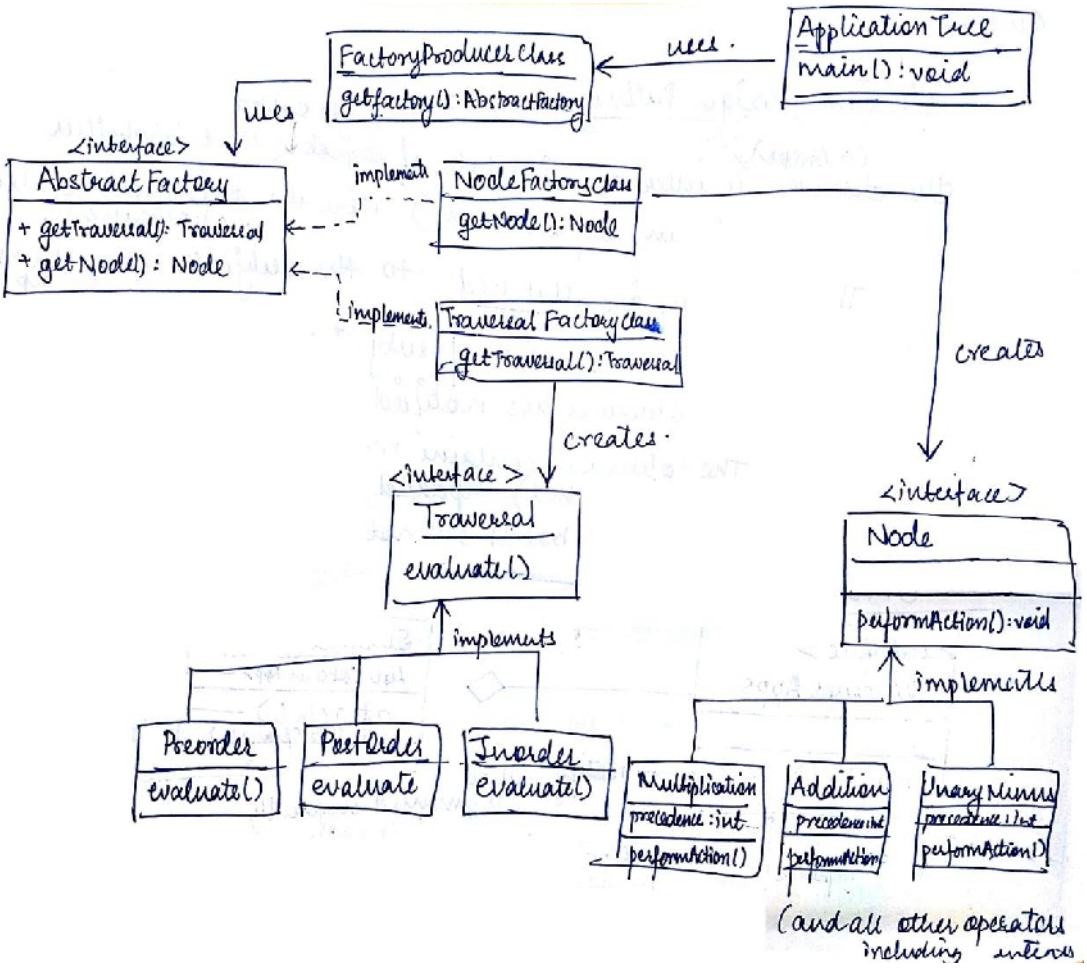
A factory created by this method could be used to create a button of either concrete type (depending on what was in the config file).

Since the structure of the tree is the same regardless of the traversal type, we can use the abstract factory pattern to create objects that traverse trees in different manners. Considering the explanation of the abstract factory pattern given above as a reference:

(a) Instead of having an interface for button, **we would have an interface for traverse, for which a** method traverse(Tree tree) would exist.
(b) The classes that inherit from this interface would be InOrderTraversal, PostOrderTraversal, PreOrderTraversal, and LevelOrderTraversal. These classes would implement the various traversals of a tree [you don't have to show in the code how traversal is done – the method stub is sufficient].
(c) We need an OrderFactory interface which would again have factories for each type of traversal that inherit from the interface, which in turn create the concrete objects required to traverse a tree in the desired way.
(d) You can assume that the tree is given to us already created and we simply pass it into the traverse method of the created Traversal.
(e) However, we can apply this pattern once more to tree creation. For example, we could have a factory for each type of node. Creation methods for addition nodes for example would take two child nodes, and the factory would return an addition node with two children which could later be added together.
(f) From here we would create the tree from the bottom up; when we have created the root node, we can pass this into the Traversal created by a factory above.

Marking scheme:

- If your solution (UML diagram and code) covers point (a), (b), and (c) as it is you will be awarded a maximum of 10 marks otherwise looking at the gravity of the mistakes you have done partial marks will be awarded.
- If your solution (UML diagram and code) covers point (e) and (f) as it is you will be awarded a maximum of 10 marks otherwise looking at the gravity of the mistakes you have done partial marks will be awarded. No marks for point (d) and for tree creation.


FactoryProducerClass
getFactory(): AbstractFactory
uses
ApplicationTree
main(): void
uses
<interface>
Abstract Factory
+ getTraversal(): Traversal
+ getNode(): Node
implements
NodeFactoryClass
getNode(): Node
implements
TraversalFactoryClass
getTraversal(): Traversal
creates
creates
<interface>
Traversal
evaluate()
implements
<interface>
Node
performAction(): void
implements
Preorder
evaluate()
PostOrder
evaluate
Inorder
evaluate()
Multiplication
precedence: int
performAction()
Addition
precedence: int
performAction
Unary Minus
precedence: int
performAction()
(and all other operators including unions

******* BEST OF LUCK *******