Student ID_____Student Name_____

Advanced Software Development DE Final Exam

October 31, 2015

PRIVATE AND CONFIDENTIAL

1. **Allotted exam duration is 2 hours.**
2. **Closed book/notes.**
3. **No personal items including electronic devices (cell phones, computers, calculators, PDAs).**
4. **Cell phones must be turned in to your proctor before beginning exam.**
5. **No additional papers are allowed. Sufficient blank paper is included in the exam packet.**
6. **Exams are copyrighted and may not be copied or transferred.**
7. **Restroom and other personal breaks are not permitted.**
8. **Total exam including questions and scratch paper must be returned to the proctor.**

4 blank pages are provided for writing the solutions and/or scratch paper. All 4 pages must be handed in with the exam

BE VERY CAREFUL WITH THE GIVEN 2 HOURS AND USE YOUR TIME WISELY. THE ALLOTED TIME IS GIVEN FOR EVERY QUESTION.

Write your name and student id at the top of this page.

Question 1 [10 points] {15 minutes}

Consider the following application that uses a dynamic proxy:

```
public interface IVehicle {
       void start();
}


public class Car implements IVehicle {
      private String name ="Herbie";

      public void start() {
            System.out.println("Car " + name + " started");
      }

}


public class Logger implements InvocationHandler {
      private Object v;

      public Logger(Object v) {
            this.v = v;
      }

      public Object invoke(Object proxy, Method m, Object[] args)
                  throws Throwable {
            System.out.println("Logger: " + m.getName());
            Object object= m.invoke(v, args);
            return object;
      }
}


public class Notifier implements InvocationHandler {
      private Object v;

      public Notifier(Object v) {
            this.v = v;
      }

      public Object invoke(Object proxy, Method m, Object[] args)
                  throws Throwable {
            System.out.println("Notifier: " + m.getName());
            Object object= m.invoke(v, args);
            System.out.println("Notifier: " + m.getName());
            return object;
      }
}
```

What is the output written to the console when we run the following application?

```
public class Application {
      public static void main(String[] args) {
            IVehicle c = new Car();
            ClassLoader cl = IVehicle.class.getClassLoader();
            IVehicle v1 = (IVehicle) Proxy.newProxyInstance(cl, new Class[]
                { IVehicle.class }, new Logger(c));
            IVehicle v2 = (IVehicle) Proxy.newProxyInstance(cl, new Class[]
                { IVehicle.class }, new Notifier(v1));
            v2.start();
      }
}
```

Your answer:

```
Notifier: start
Logger: start
Car Herbie started
Notifier: start
```

Question 2 [15 points] {15 minutes}

Consider the following given application:

```
public class Person {
      private String name;

      public Person(String name) {
            super();
            this.name = name;
      }

      public String getName() {
            return name;
      }

      public void setName(String name) {
            this.name = name;
      }

}


public interface IPersonDAO {
    Person getPerson(String ssn);
}

public class PersonDAO implements IPersonDAO{

      public Person getPerson(String ssn) {
            if (ssn.equals("324-554-5678"))
               return new Person("Frank Brown");
            if (ssn.equals("233-332-4444"))
                     return new Person("John Doe");
            return null;
      }
}


public interface IPersonService {
    public Person findPerson(String ssn);
}

public class PersonService implements IPersonService{
    private PersonDAO personDao;

      public Person findPerson(String ssn) {
            return personDao.getPerson(ssn);
      }

      public void setPersonDao(PersonDAO personDao) {
            this.personDao = personDao;
      }

}


public class Application {
      public static void main(String[] args) {
```

```
          ApplicationContext context = new
ClassPathXmlApplicationContext("springconfig.xml");
          IPersonService personService =
(IPersonService)context.getBean("personService");
          Person p = personService.findPerson("324-554-5678");
          System.out.println(p.getName());
          p = personService.findPerson("233-332-4444");
          System.out.println(p.getName());
      }

}

@Aspect
public class ServiceCounter {
      private int count=0;

      @After("execution(* PersonService.findPerson(..))")
      public void log(JoinPoint joinpoint) {
            count=count+1;
            System.out.println("method= "+
joinpoint.getSignature().getName() + " is called " + count + " times" );
      }

}

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
      http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

     <aop:aspectj-autoproxy/>
     <bean id="personService" class="PersonService">
        <property name="personDao" ref="personDao" />
    </bean>
    <bean id="personDao" class="PersonDAO"/>
    <bean id="serviceCounter" class="ServiceCounter"/>
</beans>
```

The given application is not a Spring application.
Running this application gives the following output:

```
Frank Brown
John Doe
```

The disadvantage of this simple application is that:

1. If we want to use another DAO class, we have to modify the code.
2. In the given code, the log() method of the ServiceCounter class is not called. We would like to call this method every time we call the findPerson() method on the PersonService class.

Modify the given code (**do NOT write all code again on your paper, this takes too much time**) such that:

1. The application becomes a Spring application.
2. Use Spring dependency injection to inject the PersonDAO into the PersonService. It should be possible to inject another DAO class into the PersonService without modifying any Java code.
3. Use Spring AOP for implementing the functionality that the log() method of the ServiceCounter class is called every time we call the findPerson() method on the PersonService class.

Running the Spring application should give the following output:

```
method= findCustomer is called 1 times
Frank Brown
method= findCustomer is called 2 times
John Doe
```

Question 3 [35 points] {45 minutes}

Suppose you need to design a calculator framework that allow us to calculate values like a normal calculator.

We have the following requirements for the **calculator framework**:

1. The framework should support add and subtract operations
2. The framework supports undo/redo functionality
3. It should be possible to see the list of all calculations you have done on the calculator including the current date, the calculator value (for example 12), the operation (for example "add"), the operation value (for example 5) and the result of the calculation. (in this example 17)
4. The framework will write all calculations that you do in a file on the filesystem
5. It should be easy to add other listener classes that need to do something with the calculator value when it changes.

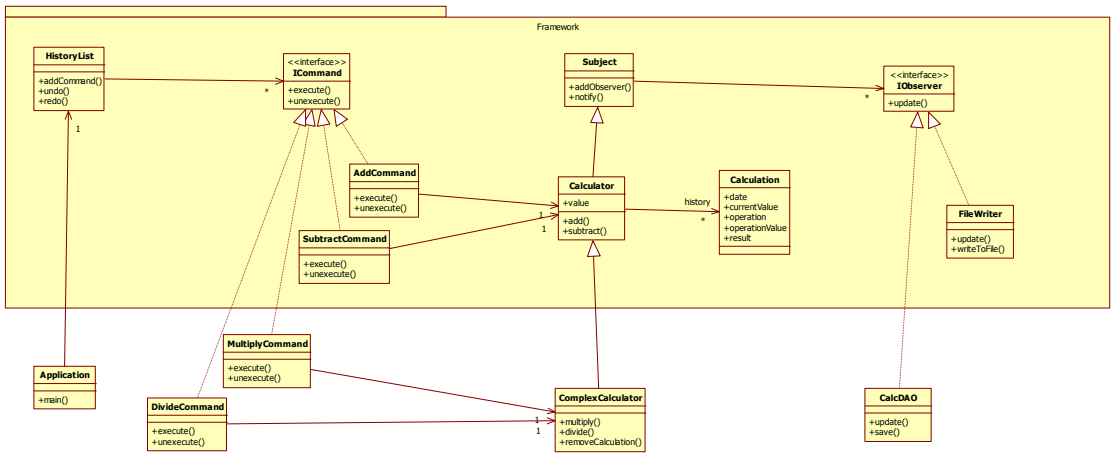
With this framework we need to design an application that uses this framework. This application has the following requirements:

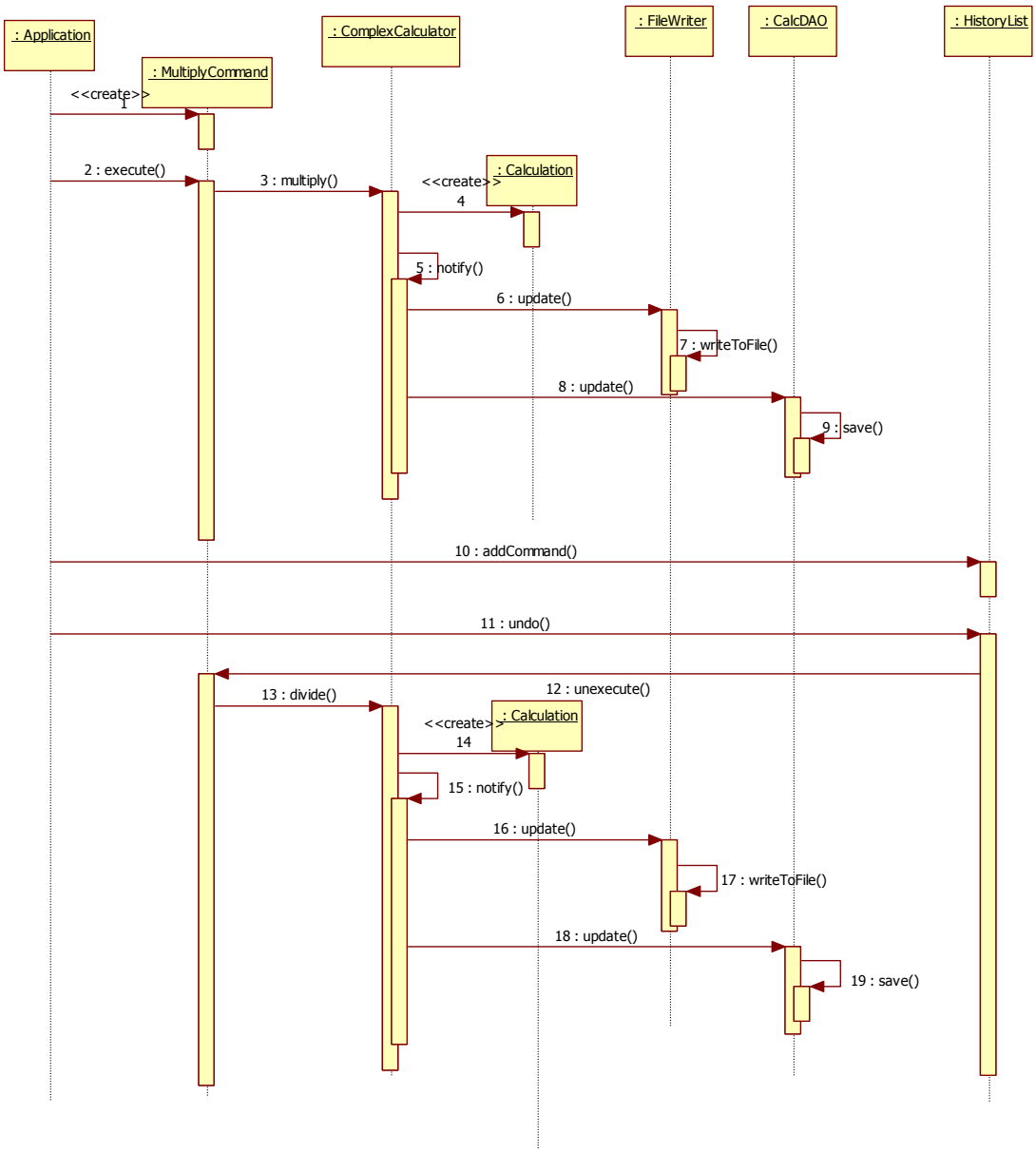
1. The application also supports multiply and divide operations (next to add and subtract operations)
2. The application will write all calculations that you do in the database.

a. Draw the class diagram of your design.
   Make sure you add all necessary UML elements (interfaces, abstract classes, attributes, methods, multiplicity, etc) to communicate the important parts of your design.
   So this class diagram should show the design of the framework, and the design of the application that uses the framework. **In the class diagram, show clearly which classes are within the framework, and which classes are outside the framework.**

   Make sure that your design follows the design principles we studied in this course.

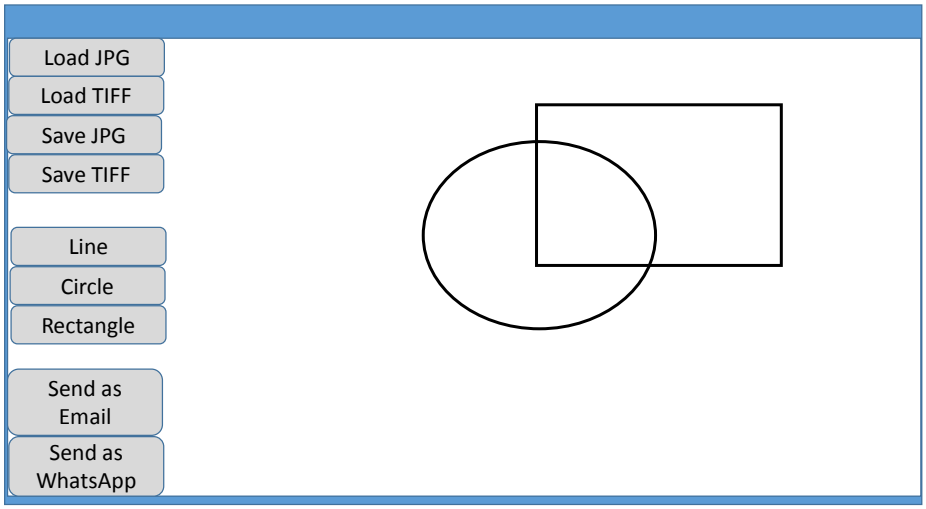
b. Draw the sequence diagram that shows how your design works. Make sure you add all necessary UML elements to communicate the important parts of your design. The sequence diagram should show the following scenario:
   1. First you do a multiplication operation with the calculator
   2. Then you do an undo action that undo's this multiplication action.


Framework
HistoryList
+addCommand()
+undo()
+redo()
<<interface>>
ICommand
+execute()
+unexecute()
Subject
+addObserver()
+notify()
<<interface>>
IObserver
+update()
AddCommand
+execute()
+unexecute()
Calculator
+value
+add()
+subtract()
history
Calculation
+date
+currentValue
+operation
+operationValue
+result
FileWriter
+update()
+writeToFile()
SubtractCommand
+execute()
+unexecute()
MultiplyCommand
+execute()
+unexecute()
Application
+main()
DivideCommand
+execute()
+unexecute()
ComplexCalculator
+multiply()
+divide()
+removeCalculation()
CalcDAO
+update()
+save()

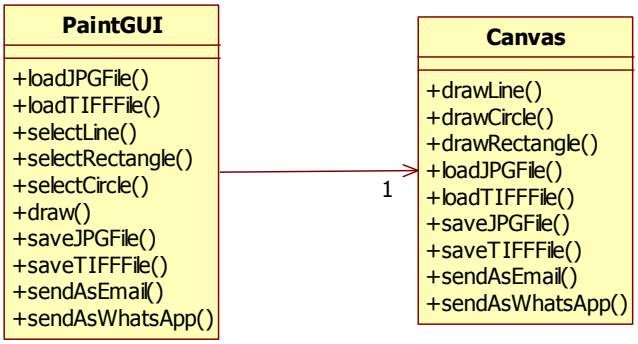

: Application
: MultiplyCommand
: ComplexCalculator
: FileWriter
: CalcDAO
: HistoryList
<<create>>
1
2 : execute()
3 : multiply()
<<create>>
: Calculation
4
5 : notify()
6 : update()
7 : writeToFile()
8 : update()
9 : save()
10 : addCommand()
11 : undo()
12 : unexecute()
13 : divide()
<<create>>
: Calculation
14
15 : notify()
16 : update()
17 : writeToFile()
18 : update()
19 : save()

Question 4 [35 points] {45 minutes}

Suppose you wrote the following paint application:



The application allows you to draw lines, circles and rectangles. If you want to draw a line, you first click the Line button and then you click with your mouse on the canvas at the position where the line should start. Then you keep the mouse button pressed and you move the mouse to the end position of the line. When you release the mouse button, the line is drawn. You can also save the drawing as a JPG picture or as a TIFF picture. You can also load a JPG or TIFF picture.
The application also allows you to send your picture by email or by WhatsApp message.
Your current application has the following design:



Now you need to design a paint framework that allows you to write these kind of paint-like applications. The framework has the following requirements:

- The framework supports opening and saving JPG and TIFF files
- Drawing lines, circles and rectangles
- It should be easy to add additional shapes
- Send the drawing by email and WhatsApp.
- It should be easy to send the drawing by another medium

With this framework we need to design an drawing application that uses this framework. This application has the following requirements (in addition to the requirements of the framework):

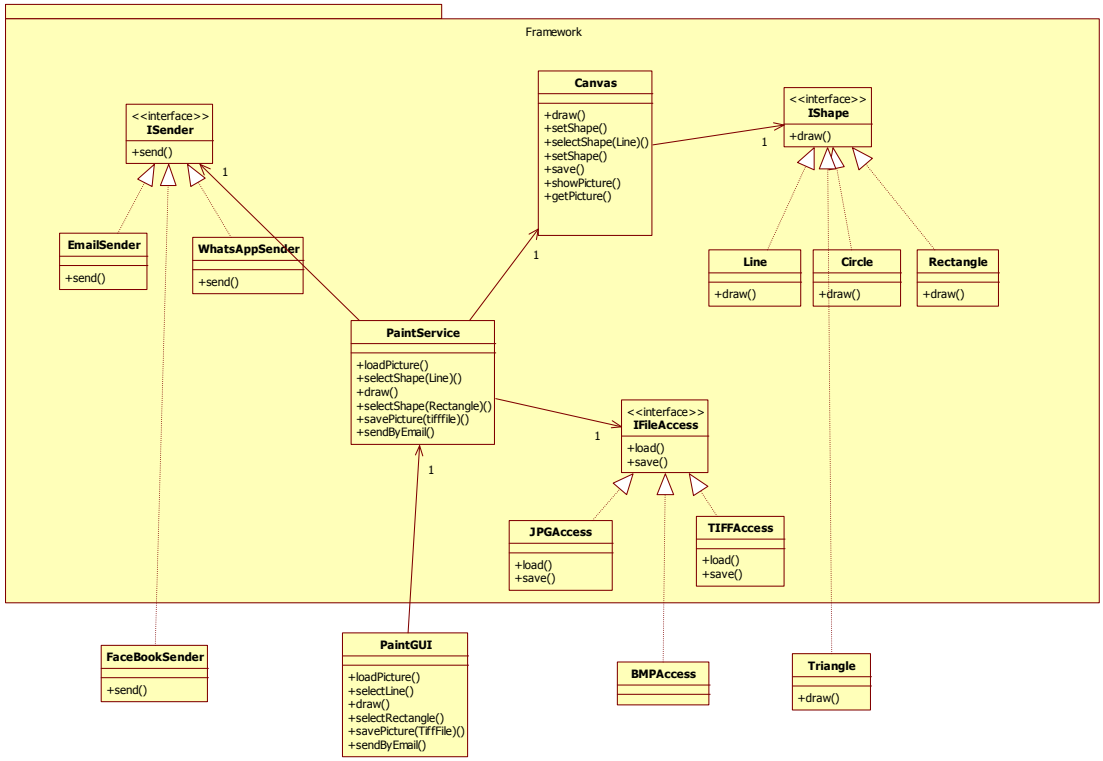
1. With the drawing application you can also draw triangles
2. With the drawing application you can also load and save BMP files
3. With the drawing application you can also send the drawing to your Facebook page

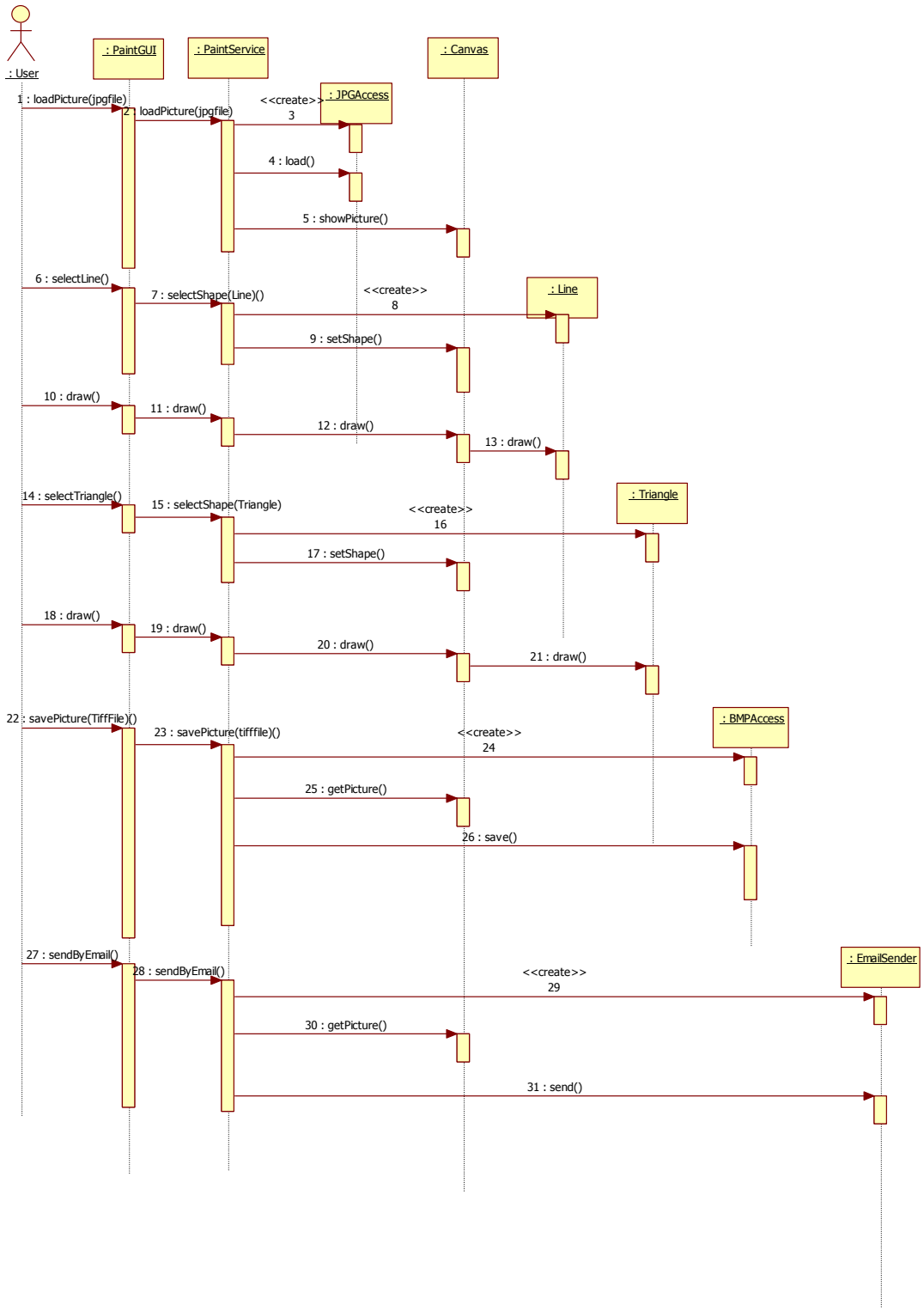
a. Draw the class diagram of your design.
   Make sure you add all necessary UML elements (interfaces, abstract classes, attributes, methods, multiplicity, etc) to communicate the important parts of your design.
   So this class diagram should show the design of the framework, and the design of the application that uses the framework. **In the class diagram, show clearly which classes are within the framework, and which classes are outside the framework.**

   Make sure that your design follows the design principles we studied in this course.

b. Draw the sequence diagram that shows how your design works. Make sure you add all necessary UML elements to communicate the important parts of your design. The sequence diagram should show the following scenario:
   1. First you load a JPG file
   2. Then you click the Line button
   3. Then you draw a line on the canvas with the mouse
   4. Then you click the Triangle button
   5. Then you draw a triangle on the canvas with the mouse
   6. Then you save the drawing as a BMP file
   7. Then send the drawing by email

   Use the partial given sequence diagram on the next page


Framework
<<interface>>
ISender
+send()
1
EmailSender
+send()
WhatsAppSender
+send()
Canvas
+draw()
+setShape()
+selectShape(Line)()
+setShape()
+save()
+showPicture()
+getPicture()
1
1
<<interface>>
IShape
+draw()
Line
+draw()
Circle
+draw()
Rectangle
+draw()
PaintService
+loadPicture()
+selectShape(Line)()
+draw()
+selectShape(Rectangle)()
+savePicture(tifffile)()
+sendByEmail()
1
1
<<interface>>
IFileAccess
+load()
+save()
JPGAccess
+load()
+save()
TIFFAccess
+load()
+save()
FaceBookSender
+send()
PaintGUI
+loadPicture()
+selectLine()
+draw()
+selectRectangle()
+savePicture(TiffFile)()
+sendByEmail()
BMPAccess
Triangle
+draw()


: User
: PaintGUI
: PaintService
: Canvas
: JPGAccess
: Line
: Triangle
: BMPAccess
: EmailSender
1 : loadPicture(jpgfile)
2 : loadPicture(jpgfile)
<<create>>
3
4 : load()
5 : showPicture()
6 : selectLine()
7 : selectShape(Line)()
<<create>>
8
9 : setShape()
10 : draw()
11 : draw()
12 : draw()
13 : draw()
14 : selectTriangle()
15 : selectShape(Triangle)
<<create>>
16
17 : setShape()
18 : draw()
19 : draw()
20 : draw()
21 : draw()
22 : savePicture(TiffFile)()
23 : savePicture(tifffile)()
<<create>>
24
25 : getPicture()
26 : save()
27 : sendByEmail()
28 : sendByEmail()
<<create>>
29
30 : getPicture()
31 : send()