Introduction to Software Engineering
Quiz 2
Chapters 2 - 5

Time: 1 hour 15 minutes

Name: _______Higgins____________ ____Timothy____________

(Print Name) (Last Name) (First Name)

1. Apply the Form Template Method Refactoring to class A.

```
/* Before Refactoring */

public class A {
 void sendMessage(Service svc) {
 svc.beginTask();
 // other common parts go here.
 svc.sendMessage();
 // other common parts go here.
 svc.endTask();
 }
 void saveMessage(Service svc) {
 svc.beginTask();
 // other common parts go here.
 svc.saveMessage();
 // other common parts go here.
 svc.endTask();
 }
 void receiveMessage(Service svc) {
 // same pattern
 svc.beginTask();
 // other common parts go here.
 svc.receiveMessage();
 // other common parts go here.
 svc.endTask();
 }
}
```

```
/* After Refactoring */

public class A {
 void sendMessage(Service svc) {
 new MessageSendTask().common(svc);  }
 void saveMessage(Service svc) {
 new MessageSaveTask().common(svc);  }
 void receiveMessage(Service svc) {  new
MessageReceivTask().common(svc);  }
 }

abstract class Template {
 void common(Service svc) {
/* Write down your answer */
      svc.beginTask();
      processMessage(svc);
      svc.endTask();

} /* Write down your answer */
abstract void processMessage(Service svc);



 }

class MessageSendTask extends Template {
void processMessage(Service svc) {
svc.sendMessage();
 }
 }

class MessageSaveTask extends Template {
void processMessage(Service svc) {
svc.saveMessage();
 }
 }

class MessageReceivTask extends Template {
void processMessage(Service svc) {
svc.saveMessage();
 }
 }
```

2. Apply Extract Method refactoring to the methods onStreetViewPanoramaReady, onStreetViewPanoramaChange, and onStreetViewPanoramaClick.

```
/* Before Refactoring */

public class MapPane extends Activity implements OnMapReadyCallback {

      void onStreetViewPanoramaReady(GoogleMap m, int value) {
             LatLng latlng = new LatLng(getLatLng());
             /* Add a draggable marker when street view is ready. */
             m.setMarkerLocationEnabled(true);
```

```
                m.initMarker(CameraUpdateFactory.newLatLngZoom(latlng, value));
                m.addMarker(new MarkerOptions());
        }

        void onStreetViewPanoramaChange(GoogleMap map, int data) {
                LatLng lat = new LatLng(getLatLng());
                /* Add a draggable marker when street view is changed. */
                map.setMarkerLocationEnabled(true);
                map.initMarker(CameraUpdateFactory.newLatLngZoom(lat, data));
                map.addMarker(new MarkerOptions());
        }

        void onStreetViewPanoramaClick(GoogleMap googleMap, int buf) {
                        LatLng lng = new LatLng(getLatLng());
                /* Add a draggable marker when a user send a click event. */
                googleMap.setMarkerLocationEnabled(true);
                googleMap.initMarker(CameraUpdateFactory.newLatLngZoom(lng, buf));
                googleMap.addMarker(new MarkerOptions());
        }

        private double getLatLng() {
                if (Util.isValidLatLng())
                        return Util.getLatLng();
                return 0;
        }
}
```

```
/* After Refactoring */

public class MapPane extends Activity implements OnMapReadyCallback {
        void onStreetViewPanoramaReady(GoogleMap m, int value) {
                LatLng latlng = new LatLng(getLatLng());

                /* Write down your answer */
                addMarker(m, value, latlng);



        }

        void onStreetViewPanoramaChange(GoogleMap map, int data) {
                LatLng lat = new LatLng(getLatLng());

                /* Write down your answer */

                addMarker(map, data, lat);
        }

        void onStreetViewPanoramaClick(GoogleMap googleMap, int buf) {
                LatLng lng = new LatLng(getLatLng());

            /* Write down your answer */

            addMarker(googleMap, buf, lng);




        }

        void addMarker(GoogleMap googleMap, int buf, LatLng lng) {
            googleMap.setMarkerLocationEnabled(true);
            googleMap.initMarker(CameraUpdateFactory.newLatLngZoom(lng, buf));
            googleMap.addMarker(new MarkerOptions());
        }

        private double getLatLng() {
            if (Util.isValidLatLng())
                return Util.getLatLng();
            return 0;
        }
}
```

3. Apply the Extract Method Refactoring to class A using *Type Parameterize*.

```
/* Before Refactoring */

public class A {
        Node m1(List<Node> nodes, String p) {
            for (Node node : nodes) { // similar part
                if (node.contains(p))
                    System.out.println(node);
            }
            // other implementations
            return null;
        }
        Edge m2(List<Edge> edges, String p) {
            for (Edge edge : edges) { // similar part
```

```
                    if (edge.contains(p))
                        System.out.println(edge);
                }
                // other implementations
                return null;
        }
}

interface IGraph {
        public boolean contains(String p);
}

class Node implements IGraph {
        String name;

        public boolean contains(String p) {
                return name.contains(p);
        }
}

class Edge implements IGraph {
        String name;

        public boolean contains(String p) {
                return name.contains(p);
        }
}
```

```
/* After Refactoring */

public class A {
        Node m1(List<Node> nodes, String p) {
        /* Write down your answer */
        extractMethod(nodes, p);



                // other implementations
                return null;
        }

        Edge m2(List<Edge> edges, String p) {
```

```
        /* Write down your answer */
        extractMethod(edges, p);


                // other implementations
                return null;
        }

        <T extends IGraph> void extractMethod(List<T> list, String str) {

        /* Write down your answer */
                for (T item : list) { // similar part
                        if (item.contains(p))
                                System.out.println(item);
                }
}

interface IGraph {
        public boolean contains(String p);
}
class Node implements IGraph {
        String name;
        public boolean contains(String p) {
                return name.contains(p);
        }
}
class Edge implements IGraph {
        String name;
        public boolean contains(String p) {
                return name.contains(p);
        }
}
```

4. Apply Pull Up Method Refactoring.

```
/* Before Refactoring */

public class Employee {
}

public class Engineer extends Employee {
            private String name;

        String getName() {
                return this.name;
        }
}

public class Salesman extends Employee {
            private String name;

        String getName() {
                return this.name;
```

```
	}
}

/* After Refactoring */

public class Employee {

	/* Write down your answer */
	String name;
	String getName() {
		return this.name;
	}



}

public class Engineer extends Employee {

}

public class Salesman extends Employee {

}
```

5. Apply Extract Superclass Refactoring.

```
/* Before Refactoring */

public class A {
 private String pkgName, className;

 public boolean visit(MethodDeclaration methodDecl) {
 String methodName = methodDecl.getName().getIdentifier();  int
parmSize = methodDecl.parameters().size();
 Type returnType = methodDecl.getReturnType2();
 boolean isRetVoid = false;
 if (returnType.isPrimitiveType()) {
 PrimitiveType pt = (PrimitiveType) returnType;
 if (pt.getPrimitiveTypeCode().equals(PrimitiveType.VOID)) {
isRetVoid = true;
 }
 }
 ModelProvider.addProgramElements(pkgName, className, methodName, //
isRetVoid, parmSize, methodDecl.parameters());  return true;
 }

 public boolean visit(PackageDeclaration pkgDecl) {
 pkgName = pkgDecl.getName().getFullyQualifiedName();
return true;
 }
}
```

```
public class B {
 private String pkgName, className;

 public boolean visit(MethodDeclaration node) {
 int pSize = node.parameters().size();
 String mName = node.getName().getIdentifier();
 Type rType = node.getReturnType2();
 boolean isRetVoid = false;
 if (rType.isPrimitiveType()) {
 PrimitiveType primType = (PrimitiveType) rType;  if
(primType.getPrimitiveTypeCode().equals(PrimitiveType.VOID)) {  isRetVoid
= true;
 }
 }
 ModelProvider.addProgramElements(pkgName, className, mName, //
isRetVoid, pSize, node.parameters());
 return true;
 }

 public boolean visit(TypeDeclaration typeDecl) {
 className = typeDecl.getName().getIdentifier();
 return true;
 }
}




/* After Refactoring */

public class A extends SuperClass {

 /* Write down your answer */

 public boolean visit(PackageDeclaration pkgDecl) {
 pkgName = pkgDecl.getName().getFullyQualifiedName();
      return true;
 }



}

public class B extends SuperClass {

 /* Write down your answer */

 public boolean visit(TypeDeclaration typeDecl) {
      className = typeDecl.getName().getIdentifier();
      return true;
 }



}
```

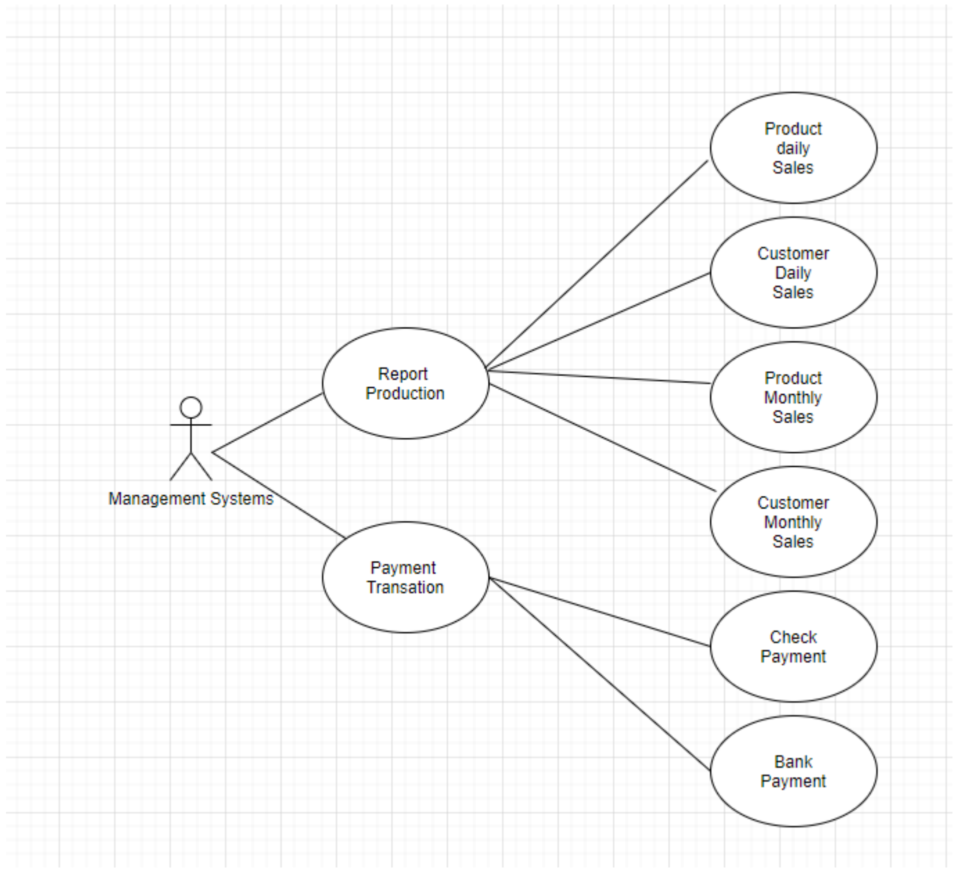
```
public class SuperClass {

 /* Write down your answer */
 protected String pkgName;
 protected String className;

 public boolean visit(MethodDeclaration methodDecl) {
 String methodName = methodDecl.getName().getIdentifier();  int
parmSize = methodDecl.parameters().size();
 Type returnType = methodDecl.getReturnType2();
 boolean isRetVoid = false;

 if (returnType.isPrimitiveType()) {
 PrimitiveType pt = (PrimitiveType) returnType;  if
(pt.getPrimitiveTypeCode().equals(PrimitiveType.VOID)) {  isRetVoid =
true;
 }
 }
 ModelProvider.addProgramElements(pkgName, className, methodName, //
isRetVoid, parmSize, methodDecl.parameters());  return true;
 }

}
```

6. Write a test case with the annotations RunWith and Parameters in JUnit. The test class name is TestService and the test method name is testGetServiceNumber.

The testing information
The test target method: int getServiceNumber(int size, int position, int option)
The input data sets: {1001, 102, 1}, {10001, 0, -1}
The expected value: 1, 0

```
import java.util.Arrays;
import java.util.Collection;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import org.junit.Assert;
import org.junit.Test;

@RunWith(Parameterized.class)
public class TestService {
 @Parameters
 public static Collection<Object[]> data() {
 return Arrays.asList(new Object[][] { //
 { 1001, 102, 1, 1 }, { 10001, 0, -1, 0 }
 });
 }
 private int mInput1;
 private int mInput2;
 private int mInput3;
 private int mExpected;
 private Service service = new Service();

 public TestService(int input1, int input2, int input3, int expected) {

       /* Write down your answer */
        this.mInput1 = input1;
        this.mInput2 = input2;
        this.mInput3 = input3;
        this.mExpected = expected;




 }

 @Test
 public void testGetServiceNumber() {
 int actualResult = service.getServiceNumber(mInput1, mInput2, mInput3);

        /* Write down your answer */
        Assert.assertEquals(mExpected, actualResult );
        }
}
```

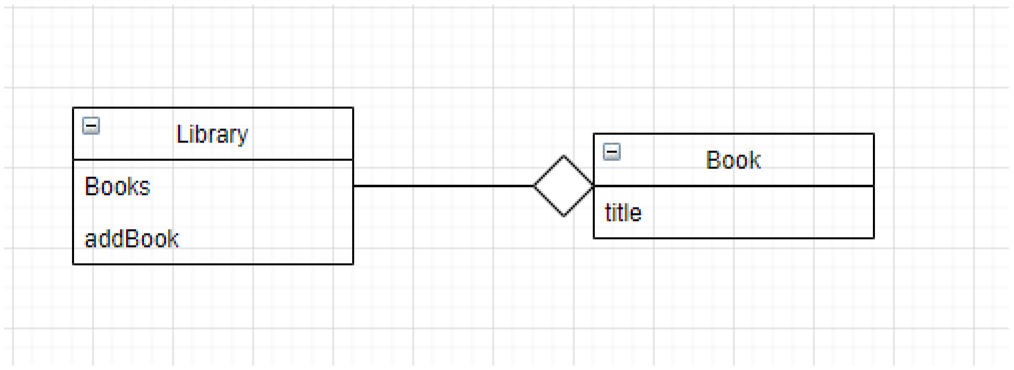
7. There are use cases "Product Daily Sales", "Customer Daily Sales", "Product Monthly Sales", and "Customer Monthly Sales" which make a use of the behavior of a use case "Report Production". Also, there are use cases "Check Payment " and "Bank Payment" which extend the behavior of a use case "Payment Transaction". A system "Management System" maintains the use cases "Report Production" and "Payment Transaction". **Draw a use case diagram to represent use cases, an actor, and the relationships among them.**



8. You have the following implementation written in Java. **Please complete a class diagram.**

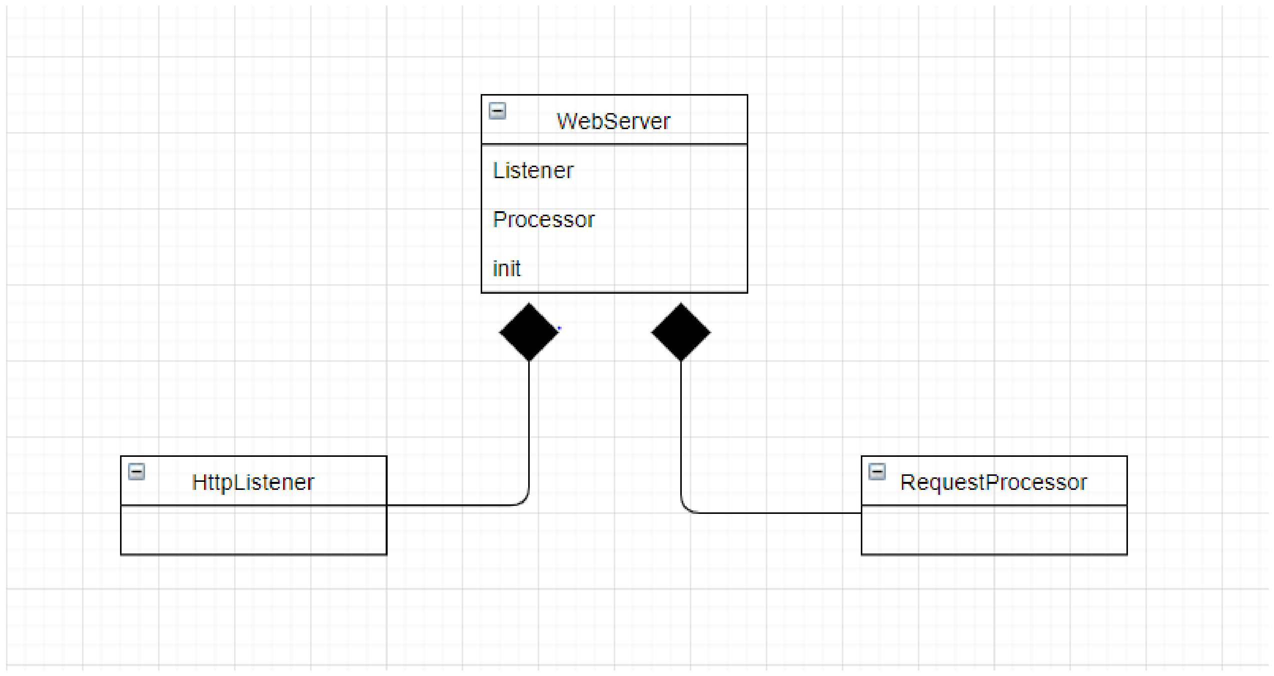
(a) Aggregation relationship

```
Public class Library {

       List<Book> books = new ArrayList<Book>();

       void addBook(Book b) {
              books.add(b);
       }
}

Public class Book {
       String title;
}
```



(b) Composition relationship

```
public class WebServer {

       Private HttpListener listener;
       Private RequestProcessor processor;

       Public init() {
             this.listener = new HttpListener(80);
             this.processor = new RequestProcessor("/www/root");
       }
}

public class HttpListener { /* ... */ }

public class RequestProcessor { /* ... */ }
```



9. Write the reasons why software engineers should perform refactorings on their program.
Improved modularity
Improved Readability
Features are easier to add
Less bugs
Improved maintainability

10. Program modularization helps software engineers separate and recombine a system's component, when a program is not cleanly decomposed from the rest of the system in design and implementation. In particular, aspect-oriented programming (AOP) provides new modularizations of software systems in order to isolate supporting functionalities from the main program's business logic. AOP allows multiple functionalities to be expressed separately and automatically unified into working systems. The example is written an AOP extension, AspectJ to add additional functionality such as logging. It helps developers add additional behavior to existing code without modifying the code itself. **Write the output of the following code.** The output of System.out.println("Hello World") is Hello World.

```
public class Program {
        public static void main(String[] args) {
                Circle circle1 = new Circle(1.0, 0, 1.0);
                Circle circle2 = new Circle(3.0, 0, 2.0);
                System.out.println(circle1.area());
                System.out.println(circle1.distance(circle2));
        }
}
```

```
class Circle {
	protected double x, y, radius;

	public Circle(double r) {
		this.x = 0;
		this.y = 0;
		this.radius = r;
	}

	public Circle(double x, double y, double r) {
		this.x = x;
		this.y = y;
		this.radius = r;
	}

	public double getX() {
		return x;
	}

	public double getY() {
		return y;
	}

	public double area() {
		return 3.14 * radius * radius;
	}

	public double distance(Circle s) {
		double dx = Math.abs(s.getX() - x);
		double dy = Math.abs(s.getY() - y);
		return Math.sqrt(dx * dx + dy * dy);
	}
}

aspect TraceAspect {
	pointcut classToTrace(): within(Circle) ;

	pointcut constructorToTrace(): classToTrace() && execution(new(..));

	pointcut methodToTrace(): classToTrace() && execution(* *(..));

	before(): constructorToTrace() {
		System.out.println(thisJoinPointStaticPart.getSignature());
		Object[] paramValues = thisJoinPoint.getArgs();
		if (paramValues != null) {
			for (Object iParam : paramValues) {
				System.out.println(iParam);
			}
		}
	}

	before(): methodToTrace() {
		System.out.println(thisJoinPointStaticPart.getSignature());
	}
}
```

```
pkg.Circle(double, double, double)
1.0
0.0
1.0
pkg.Circle(double, double, double)
3.0
0.0
2.0
double pkg.Circle.area()
3.14
double pkg.Circle.distance(Circle)
double pkg.Circle.getX()
double pkg.Circle.getY()
2.0
```