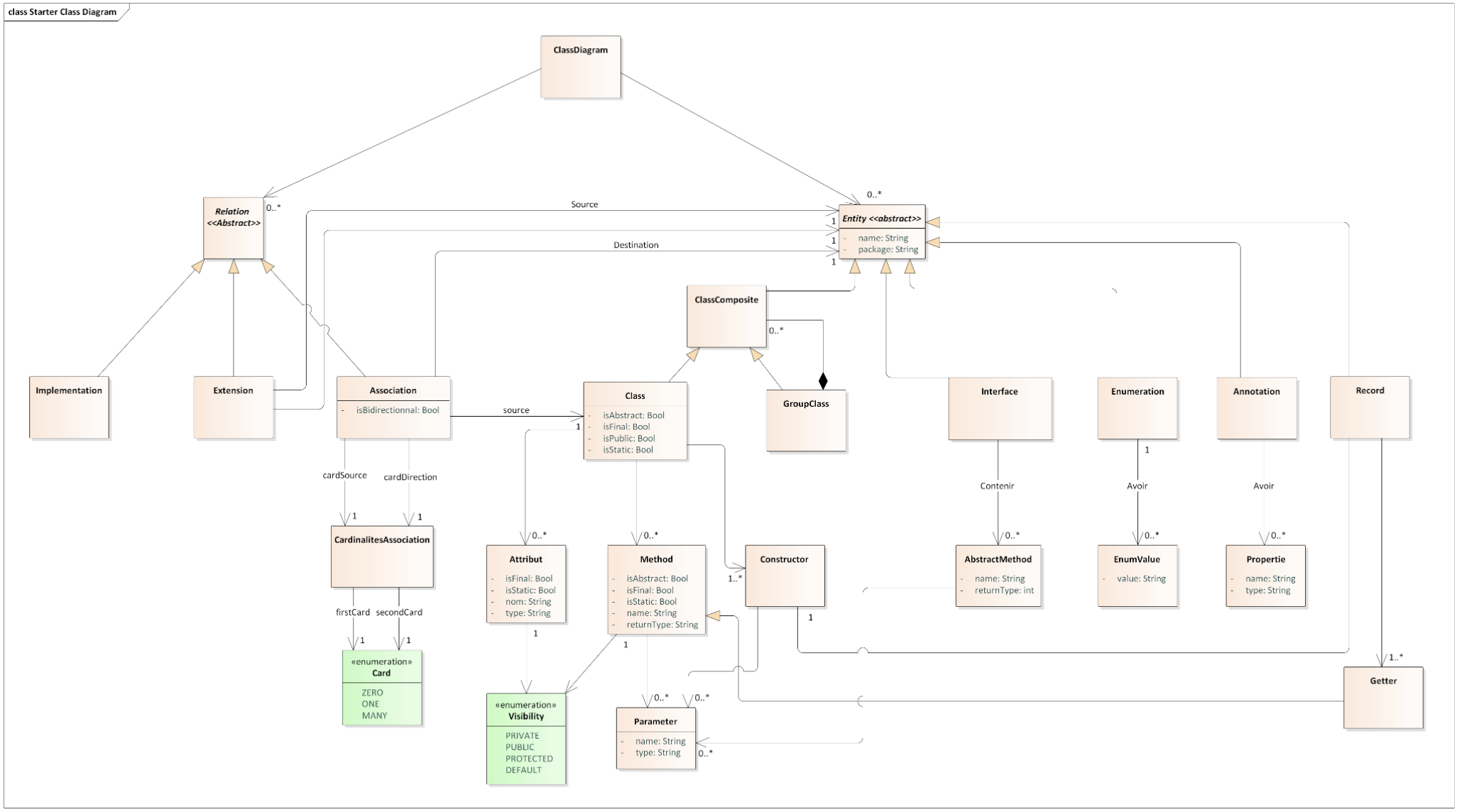
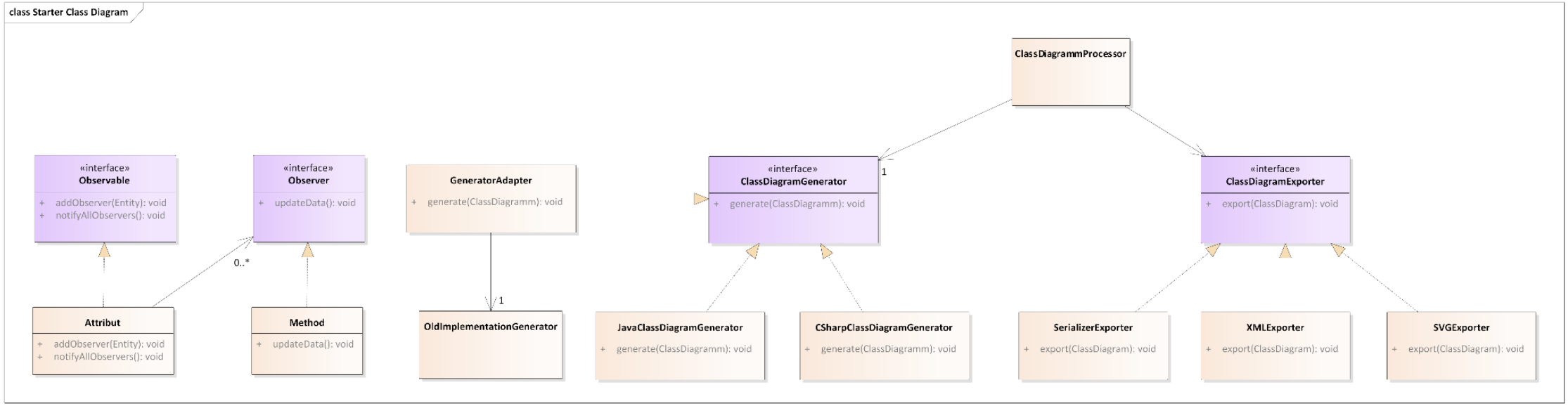
Anass Arrhioui

Diagramme de classes générale



Anass Arrhioui

Diagramme de classes pour les designs patterns



Anass Arrhioui

Diagramme de classes générale

Diagramme de classes pour les designs patterns

Les classes du diagramme

Partie Entité

Entity

```
public abstract class Entity  implements Serializable {

    protected String name;
    protected String classPackage;


    public Entity(String name, String classPackage) {
        this.name = name;
        this.classPackage = classPackage;
    }

    public String getName() {
        return name;
    }

    public String getClassPackage() {
        return classPackage;
    }
}
```

ClassComposite

```
public abstract class ClassComposite extends Entity implements Serializable {
    public ClassComposite(String name, String classPackage) {
        super(name, classPackage);
    }
}
```

Class

```
public class Class extends ClassComposite implements Serializable {

    private boolean isPublic;
    private boolean isStatic;
    private boolean isFinal;
    private boolean isAbstract;
    private List<Attribut> attributs;
    private List<Method> methods;
    private List<Constructor> constructors;




    public Class(String name, String classPackage, boolean isPublic, boolean isStatic, boolean isFinal
        super(name, classPackage);
        this.isPublic = isPublic;
        this.isStatic = isStatic;
        this.isFinal = isFinal;
        this.isAbstract = isAbstract;
        this.attributs = attributs;
        this.methods = methods;
        this.constructors = constructors;
    }

    @Override
    public String toString() {
        return "Class{" +
                ", classPackage='" + classPackage + '\'' +
                ", name='" + name + '\'' +
                "isPublic=" + isPublic +
                ", isStatic=" + isStatic +
                ", isFinal=" + isFinal +
                ", isAbstract=" + isAbstract +
                ", constructors=" + constructors +
                ", attributs=" + attributs +
                ", methods=" + methods +
                '}';
    }
}
```

GroupClass

```
public class GroupClass extends ClassComposite {
    List<ClassComposite> classComposites = new ArrayList<>();

    public GroupClass(String name, String classPackage) {
        super(name, classPackage);
    }

    public List<ClassComposite> getClassComposites() {
        return classComposites;
    }

    public void setClassComposites(List<ClassComposite> classComposites) {
        this.classComposites = classComposites;
    }
}
```

Attribut

```
public class Attribut implements Serializable, Observable {

    private String name;
    private String type;
    private boolean isStatic;
    private boolean isFinal;
    private Visibility visibility;
    private List<Observer> observers = new ArrayList<>();

    public Attribut(String name, String type, boolean isStatic, boolean isFinal, Visibility visibility
        this.name = name;
        this.type = type;
        this.isStatic = isStatic;
        this.isFinal = isFinal;
        this.visibility = visibility;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "\n\tAttribut{" +
                "name='" + name + '\'' +
                ", type='" + type + '\'' +
                ", isStatic=" + isStatic +
                ", isFinal=" + isFinal +
                ", visibility=" + visibility +
                "}\n";
    }

    @Override
    public void addObserver(Observer observer) {
        this.observers.add(observer);
    }

    @Override
    public void removeObserver(Observer observer) {
        this.observers.remove(observer);
    }

    @Override
    public void notifyAllObservers() {
        observers.forEach(observer -> observer.update(this));
    }
}
```

Method

```
public class Method implements Serializable, Observer {

    private String name;
    private String returnType;
    private boolean isStatic;
    private boolean isFinal;
    private boolean isAbstract;

    private Visibility visibility;
    private List<Parameter> parameters;


    public Method(String name, String returnType, boolean isStatic, boolean isFinal, boolean isAbstrac
        this.name = name;
        this.returnType = returnType;
        this.isStatic = isStatic;
        this.isFinal = isFinal;
        this.isAbstract = isAbstract;
        this.visibility = visibility;
        this.parameters = parameters;
    }

    @Override
    public String toString() {
        return "\n\tMethod{" +
                "name='" + name + '\'' +
                ", returnType='" + returnType + '\'' +
                ", isStatic=" + isStatic +
                ", isFinal=" + isFinal +
                ", isAbstract=" + isAbstract +
                ", visibility=" + visibility +
                ", parameters=" + parameters +
                "}\n";
    }

    @Override
    public void update(Attribut data) {
        System.out.println("Observer triggered for " + data.getName());
    }
}
```

Parameter

```
public class Parameter  implements Serializable {
    private String name;
    private String type;

    public Parameter(String name, String type) {
        this.name = name;
        this.type = type;
    }


    public String getName() {
        return name;
    }

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

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    @Override
    public String toString() {
        return "\n\t\tParameter{" +
                "name='" + name + '\'' +
                ", type='" + type + '\'' +
                "}\n";
    }
}
```

Interface

```
public class Interface extends Entity  implements Serializable {

    List<AbstractMethod> abstractMethods;

    public Interface(String name, String classPackage, List<AbstractMethod> abstractMethods) {
        super(name, classPackage);
        this.abstractMethods = abstractMethods;
    }


    @Override
    public String toString() {
        return "Interface{" +
                ", classPackage='" + classPackage + '\'' +
                ", name='" + name + '\'' +
                "abstractMethods=" + abstractMethods +
                "}\n";
    }
}
```

Record

```
public class Record extends Entity implements Serializable {

    private Constructor constructor;
    List<Attribut> attributs;

    List<Getter> getters;


    public Record(String name, String classPackage, Constructor constructor, List<Attribut> attributs,
        super(name, classPackage);
        this.constructor = constructor;
        this.attributs = attributs;
        this.getters = getters;
    }
}
```

Getter

```
public class Getter extends Method {
    public Getter(String name, String returnType, boolean isStatic, boolean isFinal, boolean isAbstrac
        super(name, returnType, false, false, false, Visibility.PUBLIC, parameters);
    }
}
```

```
public class Annotation extends Entity implements Serializable {
    public Annotation(String name, String classPackage) {
        super(name, classPackage);
    }


    @Override
    public String toString() {
        return "Annotation{" +
                "name='" + name + '\'' +
                ", classPackage='" + classPackage + '\'' +
                "}\n";
    }
}
```

```
public class EnumVal  implements Serializable {
    private String value;

    public EnumVal(String value) {
        this.value = value;
    }


    @Override
    public String toString() {
        return "EnumVal{" +
                "value='" + value + '\'' +
                "}\n";
    }
}
```

Relations

Relation

```
public abstract class Relation  implements Serializable {

}
```

Association

```
public class Association extends Relation implements Serializable {

    private Class source;
    private Class destination;
    private Cardinalite cardMinSource;
    private Cardinalite cardMaxSource;
    private Cardinalite cardMinDestination;
    private Cardinalite cardMaxDestination;


    public Association(Class source, Class destination, Cardinalite cardMinSource, Cardinalite cardMax
        this.source = source;
        this.destination = destination;
        this.cardMinSource = cardMinSource;
        this.cardMaxSource = cardMaxSource;
        this.cardMinDestination = cardMinDestination;
        this.cardMaxDestination = cardMaxDestination;
    }

    @Override
    public String toString() {
        return "\n\t\tAssociation{" +
                "source=" + source.getClassPackage()+"."+source.getName() +
                ", destination=" + destination.getClassPackage()+"."+destination.getName() +
                ", cardMinSource=" + cardMinSource +
                ", cardMaxSource=" + cardMaxSource +
                ", cardMinDestination=" + cardMinDestination +
                ", cardMaxDestination=" + cardMaxDestination +
                "}\n";
    }
}
```

CardinaliteAssociation

```
public class CardinaliteAssociation  implements Serializable {

    private Cardinalite firstCard;
    private Cardinalite secondCard;


    public CardinaliteAssociation(Cardinalite firstCard, Cardinalite secondCard) {
        this.firstCard = firstCard;
        this.secondCard = secondCard;
    }

    @Override
    public String toString() {
        return "\n\t\tCardinaliteAssociation{" +
                "firstCard=" + firstCard +
                ", secondCard=" + secondCard +
                "}\n";
    }
}
```

Cardinalite

```
public enum Cardinalite  implements Serializable {
    ZERO, ONE, MANY
}
```

Extention

```
public class Extention extends Relation implements Serializable {

    private Entity source;
    private Entity destination;

    public Extention(Entity source, Entity destination) {
        this.source = source;
        this.destination = destination;
    }

    @Override
    public String toString() {
        return "\n\tExtention{" +
                "source=" + source +
                ", destination=" + destination +
                "}\n";
    }
}
```

```
public class ClassDiagram implements Serializable {

    private List<Entity> entities = new ArrayList<>();
    private List<Relation> relations = new ArrayList<>();

    public List<Entity> getEntities() {
        return entities;
    }

    public void setEntities(List<Entity> entities) {
        this.entities = entities;
    }

    public List<Relation> getRelations() {
        return relations;
    }

    public void setRelations(List<Relation> relations) {
        this.relations = relations;
    }
}
```

Partie Design Patterns

Strategy pour la génération du code

ClassDiagramGenerator

```
public interface ClassDiagramGenerator {

    void generate(ClassDiagram classDiagram);
}
```

JavaClassDiagramGenerator

```
public class JavaClassDiagramGenerator implements ClassDiagramGenerator {

    @Override
    public void generate(ClassDiagram classDiagram) {
        System.out.println("########### Structures ###########");
        classDiagram.getEntities()
                .forEach(System.out::println);

        System.out.println("########### Relations ###########");
        classDiagram.getRelations()
                .forEach(System.out::println);
    }
}
```

Adapter de la Strategy pour la génération du code

OldGenerator

```
public class OldGenerator {

    public void generateClassDiagram(ClassDiagram classDiagram) {
        System.out.println("++++++++++++++ Structure ++++++++++++++");
        for (Entity e : classDiagram.getEntities())
            System.out.println(e);

        System.out.println("++++++++++++++ Relation ++++++++++++++");
        for (Relation relation : classDiagram.getRelations())
            System.out.println(relation);
    }
}
```

GeneratorAdapter

Implémentation basé sur la composition

```
public class GeneratorAdapter implements ClassDiagramGenerator {

    private OldGenerator oldGenerator = new OldGenerator();
    @Override
    public void generate(ClassDiagram classDiagram) {
        oldGenerator.generateClassDiagram(classDiagram);
    }
}
```

Strategy pour l'export du diagramme

ClassDiagramExporter

```
public interface ClassDiagramExporter {

    void export(ClassDiagram classDiagram, String path);
}
```

SerialiserDiagramExporter

```
public class SerialiserDiagramExporter implements ClassDiagramExporter {

    @Override
    @Lock
    public void export(ClassDiagram classDiagram, String path) {

            try(FileOutputStream fileOutputStream = new FileOutputStream(path)) {
                try (ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);
                    objectOutputStream.writeObject(classDiagram);
                }
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
    }
}
```

SVGDiagramExporter

```
public class SVGDiagramExporter implements ClassDiagramExporter {
    @Override
    public void export(ClassDiagram classDiagram, String path) {

        System.out.println("SVG export.....");
    }
}
```

XMLDiagramExporter

```
public class XMLDiagramExporter implements ClassDiagramExporter {
    @Override
    public void export(ClassDiagram classDiagram, String path) {

        System.out.println("XML export.....");
    }
}
```

Observer

Observable

```
 package me.arrhioui.observer;

public interface Observable {
    void addObserver(Observer observer);
    void removeObserver(Observer observer);
    void notifyAllObservers();
}
```

Observable

```
 package me.arrhioui.observer;

public interface Observer {
    void update(Attribut data);
}
```

Observable

```
public class Attribut implements Serializable, Observable {
    // Autres attributs
    //.....
    private List<Observer> observers = new ArrayList<>();

    // Autres Méthodes
    // .....

    @Override
    public void addObserver(Observer observer) {
        this.observers.add(observer);
    }

    @Override
    public void removeObserver(Observer observer) {
        this.observers.remove(observer);
    }

    @Override
    public void notifyAllObservers() {
        observers.forEach(observer -> observer.update(this));
    }
}
```

Observable

```
public class Method implements Serializable, Observer {

    @Override
    public void update(Attribut data) {
        System.out.println("Observer triggered for " + data.getName());
    }
}
```

Aspect

Lock

```
public @interface Lock { }
```

Log

```
public @interface Log { }
```

LogAspect

```
@Aspect
public class LogAspect {

    @Before(value = "@annotation(me.arrhioui.aspect.Log)")
    public void log(JoinPoint joinPoint){
        System.out.println("Execution of " + joinPoint.getSignature().getName());
    }
}
```

LockAspect

```
@Aspect
public class LockAspect {

    @Around(value = "@annotation(me.arrhioui.aspect.Lock)")
    public Object lockMethod(ProceedingJoinPoint pjp){
        System.out.println("The method " + pjp.getSignature() +", is Locked");
        return null;
    }
}
```