ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Département Mathématique et Informatique
Génie du Logiciel et des Systèmes Informatique Distribués

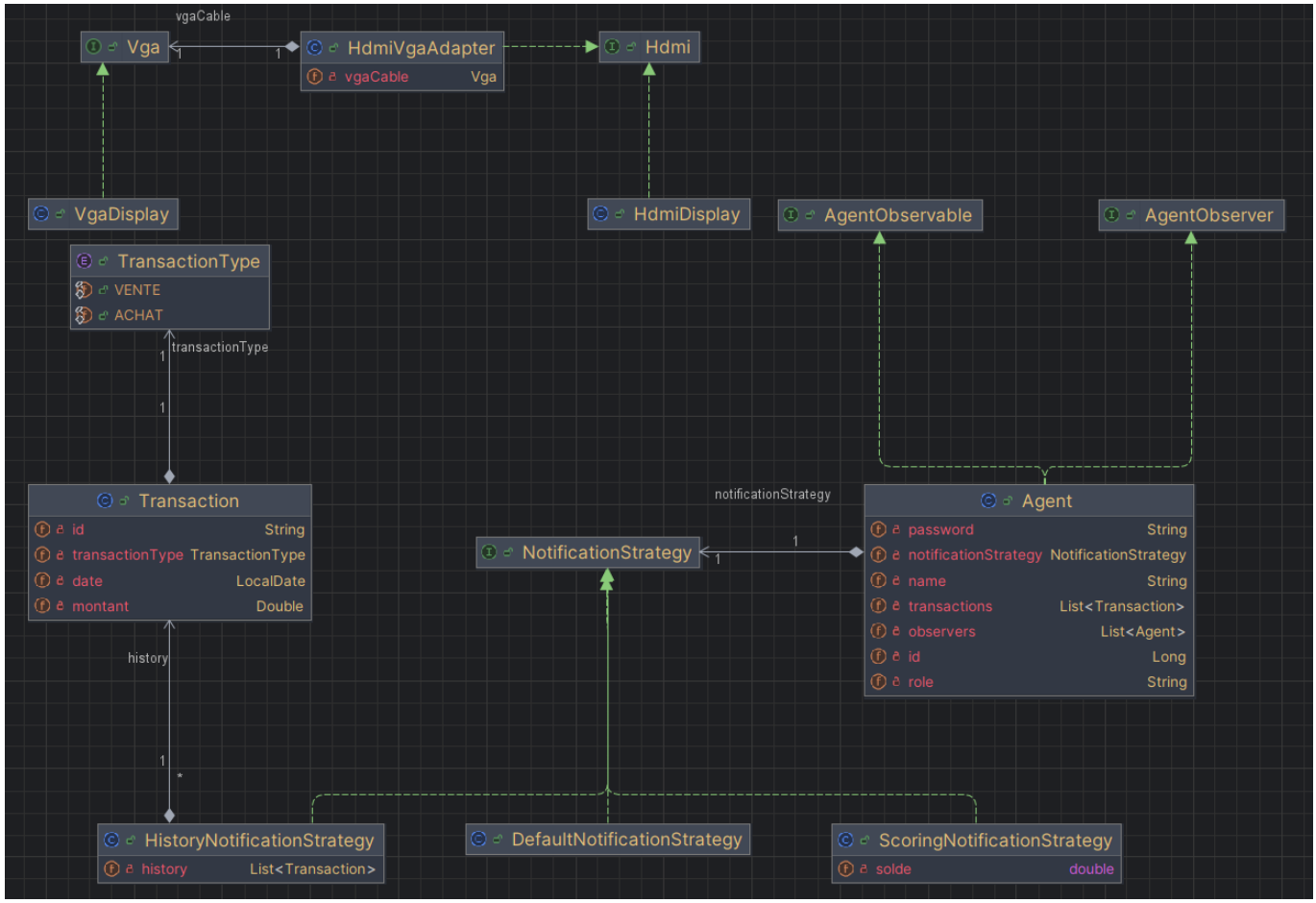
RAPPORT EXAM BLANC
MODULE: Design Pattern

Filière: ***Génie du Logiciel et des Systèmes Informatique Distribués | Semestre 5***

Réalisé par:
Achraf HAMMI
CNE : M130014277

Enseigné par :
Pr. Mohamed EL YOUSSFI

1. Diagramme de classe :



2. Implémenter et tester la classe Transaction :

```java
package achraf.entities;

import java.time.LocalDate;

public class Transaction {  26 usages  new *
    private String id;  3 usages
    private LocalDate date;  3 usages
    private Double montant;  3 usages
    private TransactionType transactionType;  3 usages

    public Transaction(Builder builder) {  1 usage  new *
        this.id = builder.id;
        this.date = builder.date;
        this.montant = builder.montant;
        this.transactionType = builder.transactionType;
    }

    public String getId() {  no usages  new *
        return id;
    }

    public LocalDate getLocalDate() {  no usages  new *
        return date;
    }

    public Double getMontant() {  3 usages  new *
        return montant;
```

```java
    public TransactionType getTransactionType() { 1 usage  new *
        return transactionType;
    }


    public static Builder builder() { 1 usage  new *
        return new Builder();
    }

    public static class Builder { 7 usages  new *
        private String id; 2 usages
        private LocalDate date; 2 usages
        private double montant; 2 usages
        private TransactionType transactionType; 2 usages

        public Builder id(String id) { no usages  new *
            this.id = id;
            return this;
        }

        public Builder date(LocalDate date) { no usages  new *
            this.date = date;
            return this;
        }

        public Builder montant(double montant) { no usages  new *
            this.montant = montant;
            return this;
        }

        public Builder transactionType(TransactionType transactionType) { no usages  new *
            this.transactionType = transactionType;
            return this;
        }

        public Transaction build() { 1 usage  new *
            return new Transaction( builder: this);
        }
    }

    @Override  new *
    public String toString() {
        return "Transaction " + id + " [" + date + ", " + montant + ", " + transactionType + "]";
    }
}
```

- Test dans main :

```java
private static void testTransactionBuilder() { 1 usage  achrafhammi *
    System.out.println("=== test de builder ===");
    Transaction transaction = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(1000.0)
            .build();

    System.out.println("transaction crée: " + transaction);
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Users\Achraf HAMMI
=== test de builder ===
transaction crée: Transaction 01 [2024-12-13, 1000.0, null]
```

3. Implémenter et tester la classe Agent

- ***Pour implémenter la classe Agent et ses fonctionnalités, on doit d'abord créer les interfaces de Observable et Observer selon le pattern de Observer :***

```java
package achraf.observer;

import achraf.entities.Transaction;

public interface AgentObserver { 2 usages 1 implementation achrafhammi
    void update(String agentName, Transaction transaction); 1 usage 1 implementation achrafhammi
}
```

```java
package achraf.observer;

import achraf.entities.Agent;
import achraf.entities.Transaction;

public interface AgentObservable { 2 usages 1 implementation achrafhammi
    void subscribe(Agent observer); no usages 1 implementation achrafhammi
    void unsubscribe(Agent observer); no usages 1 implementation achrafhammi
    void notifyObservers(Transaction transaction); 1 usage 1 implementation achrafhammi
}
```

- La classe Agent :

```java
public class Agent implements AgentObservable, AgentObserver { 17 usages achrafhammi *
    private Long id; 1 usage
    private String name; 4 usages
    private String role; 3 usages
    private String password; 3 usages
    private List<Agent> observers = new ArrayList<>(); 3 usages
    private List<Transaction> transactions = new ArrayList<>(); 3 usages
    private NotificationStrategy notificationStrategy = new DefaultNotificationStrategy(); no usages

    public Agent(Long id, String name, String role, String password) { 4 usages new *
        this.id = id;
        this.name = name;
        this.role = role;
        this.password = password;
    }

    public String getName() { achrafhammi
        return "Nom d'agent " + name;
    }

    @SecuredBy(roles = "USER") no usages achrafhammi
    public void showAllTransactions() {
        System.out.println("Les transactions de l'agent " + name + ": ");
        for (Transaction transaction : transactions) {
```

```
        for (Transaction transaction : transactions) {
            System.out.println(transaction);
        }
    }

    @SecuredBy(roles = "ADMIN")  no usages  achrafhammi
    public Transaction getLargestTransaction() {
        return transactions.stream() Stream<Transaction>
                .max(Comparator.comparingDouble(Transaction::getMontant)) Optional<Transaction>
                .orElse( other: null);
    }

    public String getPassword() {  1 usage  new *
        return password;
    }

    @SecuredBy(roles = "ADMIN")  no usages  achrafhammi
    public void addTransaction(Transaction transaction){
        transactions.add(transaction);
        notifyObservers(transaction);
    }
```

```
    @Override  no usages  achrafhammi
    public void subscribe(Agent agentObserver) {
        observers.add(agentObserver);
    }

    @Override  no usages  achrafhammi
    public void unsubscribe(Agent agentObserver) {
        observers.remove(agentObserver);

    }

    @Override  1 usage  achrafhammi
    public void notifyObservers(Transaction transaction) {
        for(Agent a : observers){
            a.update(a.name, transaction);
        }
    }

    @Override  1 usage  achrafhammi
    public void update(String agentName, Transaction transaction) {
        System.out.println(agentName + "a fait une transaction: "+transaction);
    }
}
```

- Test dans la fonction main :

```java
private static void testAgentOperations() {  1 usage  new *
    System.out.println("=== test de les operations d'agent ===");
    Agent agent = new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "1234");
    Transaction t1 = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(1000.0)
            .transactionType(VENTE)
            .build();
    Transaction t2 = Transaction.builder()
            .id("02")
            .date(LocalDate.now().plusWeeks( weeksToAdd: 5))
            .montant(120)
            .transactionType(ACHAT)
            .build();

    agent.addTransaction(t1);
    agent.addTransaction(t2);

    System.out.println(agent.getName());
    agent.showAllTransactions();
    Transaction maxTransaction = agent.getLargestTransaction();
    System.out.println("Max Montant Transaction: " + maxTransaction);
}
```

```java
private static void testAgentSubscription() {  1 usage  new *
    System.out.println("\n=== Testing observer pattern sur agent ===");
    Agent agent1 = new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "USER");
    Agent agent2 = new Agent( id: 2L,  name: "rachid",  password: "1234",  role: "ADMIN");
    Agent agent3 = new Agent( id: 4L,  name: "mohammed",  password: "1234",  role: "USER");

    agent1.subscribe(agent2);
    agent1.subscribe(agent3);

    Transaction transaction = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(1000.0)
            .transactionType(VENTE)
            .build();

    agent1.addTransaction(transaction);
    agent1.unsubscribe(agent3);
    System.out.println("on supprime mohammed de la liste des observers");
    agent1.addTransaction(transaction);
}
```

```
=== test de les operations d'agent ===
Nom d'agent achraf
Les transactions de l'agent achraf:
Transaction 01 [2024-12-13, 1000.0, VENTE]
Transaction 02 [2025-01-17, 120.0, ACHAT]
Max Montant Transaction: Transaction 01 [2024-12-13, 1000.0, VENTE]
=== Testing observer pattern sur agent ===
notification pour: rachid | achraf a fait une transaction: Transaction 01 [2024-12-13, 1000.0, VENTE]
notification pour: mohammed | achraf a fait une transaction: Transaction 01 [2024-12-13, 1000.0, VENTE]
on supprime mohammed de la liste des observers
notification pour: rachid | achraf a fait une transaction: Transaction 01 [2024-12-13, 1000.0, VENTE]
```

- ***Implémentation de stratégie design pattern :***

```java
package achraf.strategy;

import achraf.entities.Transaction;

public interface NotificationStrategy {
    void processNotification(String agentObserver, Transaction transaction);
}
```

```java
package achraf.strategy;

import achraf.entities.Transaction;

public class DefaultNotificationStrategy implements NotificationStrategy {
    @Override
    public void processNotification(String agentName, Transaction transaction) {
        System.out.println("NOTIFICATION! "+agentName + " a fait la transaction "+ transaction);
    }
}
```

```java
package achraf.strategy;

import achraf.entities.Transaction;
import static achraf.entities.TransactionType.VENTE;

public class ScoringNotificationStrategy implements NotificationStrategy {
    private double solde = 0;
    @Override
    public void processNotification(String agentObserver, Transaction transaction) {
        if(transaction.getTransactionType().equals(VENTE)){
            solde+=transaction.getMontant();
        }else{
            solde-=transaction.getMontant();
        }
        System.out.println("Nouveau balance: "+ solde);
    }
    public double getSolde() {
        return solde;
    }
}
```

```java
package achraf.strategy;

import achraf.entities.Transaction;

import java.util.ArrayList;
import java.util.List;

public class HistoryNotificationStrategy implements NotificationStrategy {
    private List<Transaction> history = new ArrayList<>();
    @Override
    public void processNotification(String agentObserver, Transaction transaction) {
        history.add(transaction);
        System.out.println(transaction + " crée par "+ agentObserver + " est ajouté a l'historique");
    }

    public List<Transaction> getHistory() {
        return history;
    }
}
```

- ***Test main :***

```java
private static void testNotificationStrategies() {  no usages  new *
    System.out.println("=== Testing Notification Strategies ===");
    Agent agent = new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "USER");
    Transaction transaction1 = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(1000.0)
            .transactionType(VENTE)
            .build();
    Transaction transaction2 = Transaction.builder()
            .id("02")
            .date(LocalDate.now().plusWeeks( weeksToAdd: 5))
            .montant(120)
            .transactionType(ACHAT)
            .build();
    // Default strategy
    agent.addTransaction(transaction1);
    // Scoring Strategy
    agent.setNotificationStrategy(new ScoringNotificationStrategy());
    agent.addTransaction(transaction1);
    agent.addTransaction(transaction2);
    // History Strategy
    agent.setNotificationStrategy(new HistoryNotificationStrategy());
    agent.addTransaction(transaction1);
    agent.addTransaction(transaction2);
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Users\Achraf HAMMI\AppData\Local\Pr
=== Testing Notification Strategies ===
NOTIFICATION! achraf a fait la transaction Transaction 01 [2024-12-13, 1000.0, VENTE]
Nouveau balance: 1000.0
Nouveau balance: 880.0
Transaction 01 [2024-12-13, 1000.0, VENTE] crée par achraf est ajouté a l'historique
Transaction 02 [2025-01-17, 120.0, ACHAT] crée par achraf est ajouté a l'historique
```

4. Implémenter et tester la classe Container

```java
package achraf.entities;

import achraf.monitors.Hdmi;
import java.util.HashMap;

public class Container {  5 usages  achrafhammi *
    private static Container instance;  3 usages
    private HashMap<String, Agent> agents;  6 usages
    private Hdmi hdmiCable;  6 usages

    private Container() {  1 usage  achrafhammi
        agents = new HashMap<>();
    }

    public static Container getInstance() {  1 usage  achrafhammi *
        if (instance == null) {
            instance = new Container();
        }
        return instance;
    }

    public void setHdmiCable(Hdmi hdmiCable) {  4 usages  new *
        this.hdmiCable = hdmiCable;
    }

    public void addAgent(String name, Agent agent) {  2 usages  achrafhammi *
        agents.put(name, agent);
        hdmiCable.showMessage("Agent " + name + " ajouté.");
    }

    public void removeAgent(String name) {  1 usage  achrafhammi *
        if (agents.remove(name) != null) {
            hdmiCable.showMessage("Agent " + name + " supprimé.");
        } else {
            hdmiCable.showMessage("Agent " + name + " introuvable.");
        }
    }

    public Agent getAgent(String name) {  1 usage  achrafhammi *
        if(agents.containsKey(name)){
            hdmiCable.showMessage("Agent " + name + " trouvé.");
            return agents.get(name);
        }
        hdmiCable.showMessage("Agent " + name + " introuvable.");
        return agents.get(null);
    }
}
```

- ***Adapteur design pattern***

```java
package achraf.monitors;

public interface Hdmi {  9 usages  2 implementations  achrafhammi
    void showMessage(String message);  5 usages  2 implementations  achrafhammi
}
```

```java
package achraf.monitors;

public class HdmiDisplay implements Hdmi{  2 usages  new *
    @Override  5 usages  new *
    public void showMessage(String message) {
        System.out.println("hdmi monitor:");
        System.out.println(message);
    }
}
```

```java
package achraf.monitors;

public interface Vga {  3 usages  1 implementation  achrafhammi
    void showMessage(String message);  1 usage  1 implementation  achrafhammi
}
```

```java
package achraf.monitors;

public class VgaDisplay implements Vga{  4 usages  new *
    @Override  1 usage  new *
    public void showMessage(String message) {
        System.out.println("vga monitor: ");
        System.out.println(message);
    }
}
```

```java
package achraf.adapter;

import achraf.monitors.Hdmi;
import achraf.monitors.Vga;
import achraf.monitors.VgaDisplay;

public class HdmiVgaAdapter implements Hdmi {  2 usages  new *
    private Vga vgaCable;  2 usages

    public HdmiVgaAdapter(VgaDisplay vgaDisplay) {  1 usage  new *
        vgaCable = vgaDisplay;
    }

    @Override  5 usages  new *
    public void showMessage(String message) {
        System.out.println("passe par adapter vers vga monitor: ");
        vgaCable.showMessage(message);
    }
}
```

- ***Test main :***

```
private static void testAgentContainer() {  1 usage  new *
    System.out.println("=== Testing Container ===");
    Container container = Container.getInstance();
    Hdmi hdmiDisplay = new HdmiDisplay();
    Hdmi vgaAdapter = new HdmiVgaAdapter(new VgaDisplay());

    Agent agent1 = new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "USER");
    Agent agent2 = new Agent( id: 2L,  name: "rachid",  password: "1234",  role: "ADMIN");

    container.setHdmiCable(hdmiDisplay);
    container.addAgent(agent1.getName(), agent1);
    container.setHdmiCable(vgaAdapter);
    container.addAgent(agent2.getName(), agent2);

    container.setHdmiCable(hdmiDisplay);
    Agent retrievedAgent = container.getAgent( name: "achraf");
    container.setHdmiCable(vgaAdapter);
    container.removeAgent( name: "rachid");
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Users\Achraf HAMMI\AppData\Local\Programs\IntelliJ
=== Testing Container ===
hdmi monitor:
Agent achraf ajouté.

passe par adapter vers vga monitor:
vga monitor:
Agent rachid ajouté.

hdmi monitor:
Agent achraf trouvé.

passe par adapter vers vga monitor:
vga monitor:
Agent rachid supprimé.
```

5. Implémenter les aspect techniques suivants :

- ***Un aspect de journalisation basé sur une annotation @Log à créer.***

```
package achraf.aspectj.logs;

import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Retention(RUNTIME)  no usages  new *
@Target(METHOD)
public @interface Log {
}
```

```java
package achraf.aspectj.logs;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.reflect.MethodSignature;

@Aspect new *
public class LoggingAspect {
    @Around("@annotation(Log)") new *
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {
        MethodSignature methodSignature = (MethodSignature) joinPoint.getSignature();
        String methodName = methodSignature.getMethod().getName();

        long startTime = System.nanoTime();
        Object result = joinPoint.proceed();
        long endTime = System.nanoTime();

        long executionTime = endTime - startTime;
        System.out.println("Méthode "+methodName+" a pris "+ executionTime+" nanosecondes à s'exécuter.");

        return result;
    }
}
```

- ***Test main :***

```java
@Log 1 usage  achrafhammi
public Transaction getLargestTransaction() {
    return transactions.stream() Stream<Transaction>
            .max(Comparator.comparingDouble(Transaction::getMontant)) Optional<Transaction>
            .orElse( other: null);
}

@Log 10 usages  achrafhammi *
public void addTransaction(Transaction transaction){
    transactions.add(transaction);
    notificationStrategy.processNotification(name, transaction);
    notifyObservers(transaction);
}
```

```java
private static void testLoggingAspect() {  no usages  new *
    System.out.println("=== Testing Logging Aspect ===");
    Agent agent1 = new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "USER");
    Transaction transaction1 = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(1000.0)
            .transactionType(VENTE)
            .build();
    Transaction transaction2 = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(510.0)
            .transactionType(ACHAT)
            .build();
    // method a l'annotation de log
    agent1.addTransaction(transaction1);
    agent1.addTransaction(transaction2);
    System.out.println("largest transaction: "+agent1.getLargestTransaction());
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Users\Achraf HAMMI\AppData\Local\Programs
=== Testing Logging Aspect ===
NOTIFICATION! achraf a fait la transaction Transaction 01 [2024-12-13, 1000.0, VENTE]
Méthode addTransaction a pris 1330900 nanosecondes à s'exécuter.
Méthode addTransaction a pris 1440800 nanosecondes à s'exécuter.
NOTIFICATION! achraf a fait la transaction Transaction 01 [2024-12-13, 510.0, ACHAT]
Méthode addTransaction a pris 59400 nanosecondes à s'exécuter.
Méthode addTransaction a pris 88200 nanosecondes à s'exécuter.
Méthode getLargestTransaction a pris 3317300 nanosecondes à s'exécuter.
Méthode getLargestTransaction a pris 3387000 nanosecondes à s'exécuter.
largest transaction: Transaction 01 [2024-12-13, 1000.0, VENTE]

Process finished with exit code 0
```

- ***Un aspect qui permet de définir un cache basé sur une annotation @Cachable.***

```java
package achraf.aspectj.cachable;

import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Retention(RUNTIME) 1 usage  new *
@Target(METHOD)
public @interface Cachable {
}
```

```java
@Aspect  new *
public class CachableAspect {
    private Map<String, Object> cache = new HashMap<>();  3 usages
    @Around("@annotation(Cachable)")  new *
    public Object cacheResult(ProceedingJoinPoint joinPoint) throws Throwable {
        MethodSignature methodSignature = (MethodSignature) joinPoint.getSignature();
        String methodName = methodSignature.getMethod().getName();
        Object[] arguments = joinPoint.getArgs();
        String cacheKey = generateCacheKey(methodName, arguments);
        if (cache.containsKey(cacheKey)) {
            return cache.get(cacheKey);
        }
        Object result = joinPoint.proceed();
        cache.put(cacheKey, result);
        return result;
    }
    private String generateCacheKey(String methodName, Object[] args) {  1 usage  new *
        StringBuilder key = new StringBuilder(methodName);
        for (Object arg : args) {
            key.append("|").append(arg);
        }
        return key.toString();
    }
}
```

- ***Test main :***

```java
@Cachable  4 usages  achrafhammi
public Transaction getLargestTransaction() {
    return transactions.stream() Stream<Transaction>
            .max(Comparator.comparingDouble(Transaction::getMontant)) Optional<Transaction>
            .orElse( other: null);
}
```

```java
private static void testCachingAspect() {  1 usage  new *
    System.out.println("=== Testing Caching Aspect ===");
    Agent agent1 = new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "USER");
    Transaction transaction1 = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(1000.0)
            .transactionType(VENTE)
            .build();
    Transaction transaction2 = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(510.0)
            .transactionType(ACHAT)
            .build();
    agent1.addTransaction(transaction1);
    agent1.addTransaction(transaction2);
    // il va calculer premier fois, puis deuxieme fois il utilise cache
    Transaction maxTransaction1 = agent1.getLargestTransaction();
    Transaction maxTransaction2 = agent1.getLargestTransaction();

    System.out.println("Max Transaction: " + maxTransaction1);
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Users\Achraf HAMMI\AppData\Loc
=== Testing Caching Aspect ===
NOTIFICATION! achraf a fait la transaction Transaction 01 [2024-12-13, 1000.0, VENTE]
NOTIFICATION! achraf a fait la transaction Transaction 01 [2024-12-13, 510.0, ACHAT]
nouvelle ops, donc cache vide
nouvelle ops, donc cache vide
usage de cache
Max Transaction: Transaction 01 [2024-12-13, 1000.0, VENTE]
```

- ***Un aspect qui permet de sécuriser l'application avec un username et un mot de passe et avec des roles.***

```java
@Retention(RUNTIME)  3 usages  new *
@Target(METHOD)
public @interface SecuredBy {
    String[] roles() default {};  2 usages  new *
}
```

```java
class SecurityContext {  5 usages  new *
    private static SecurityContext instance;  3 usages
    private String currentUsername;  3 usages
    private Set<String> currentUserRoles;  6 usages
    private List<Agent> usersDb;  5 usages

    private SecurityContext() {  1 usage  new *
        usersDb.add(new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "USER"));
        usersDb.add(new Agent( id: 2L,  name: "rachid",  password: "1234",  role: "ADMIN"));
        usersDb.add(new Agent( id: 4L,  name: "mohammed",  password: "1234",  role: "USER"));
        usersDb.add(new Agent( id: 5L,  name: "abdelmoula",  password: "1234",  role: "ADMIN"));
    }

    public static SecurityContext getInstance() {  1 usage  new *
        if (instance == null) {
            instance = new SecurityContext();
        }
        return instance;
    }
```

```java
    public static SecurityContext getInstance() {  1 usage  new *
        if (instance == null) {
            instance = new SecurityContext();
        }
        return instance;
    }

    public void login(String username, String password, Set<String> roles) {  no usages  new *
        for(Agent a:usersDb){
            if(Objects.equals(a.getName(), username) && Objects.equals(a.getPassword(), password)){
                this.currentUsername = username;
                this.currentUserRoles = roles;
                return;
            }
        }
        System.out.println("user n'existe pas!!!!");
    }

    public void logout() {  no usages  new *
        this.currentUsername = null;
        this.currentUserRoles = null;
    }
```

```
    public boolean hasRole(String role) {  no usages  new *
        return currentUserRoles != null && currentUserRoles.contains(role);
    }

    public boolean hasAnyRole(String[] requiredRoles) {  1 usage  new *
        if (currentUserRoles == null) return false;
        return Arrays.stream(requiredRoles)
                .anyMatch(currentUserRoles::contains);
    }

    public String getCurrentUsername() {  1 usage  new *
        return currentUsername;
    }
}
```

```
@Aspect  new *
public class SecurityAspect {

    @Around("@annotation(securedBy)")  new *
    public Object checkSecurity(ProceedingJoinPoint joinPoint, SecuredBy securedBy) throws Throwable {
        String[] requiredRoles = securedBy.roles();
        if (requiredRoles.length == 0) {
            return joinPoint.proceed();
        }
        SecurityContext securityContext = SecurityContext.getInstance();
        if (securityContext.getCurrentUsername() == null) {
            throw new SecurityException("User is not authenticated");
        }
        if (!securityContext.hasAnyRole(requiredRoles)) {
            throw new SecurityException("User does not have required roles. Required: " +
                    Arrays.toString(requiredRoles));
        }
        return joinPoint.proceed();
    }
}
```

- ***Test main :***

```
@SecuredBy(roles = "ADMIN")  15 usages  achrafhammi *
public void addTransaction(Transaction transaction){
    transactions.add(transaction);
    notificationStrategy.processNotification(name, transaction);
    notifyObservers(transaction);
}
```

```java
private static void testSecurityAspect() {  1 usage  new *
    System.out.println("=== Testing Security Aspect ===");
    SecurityContext securityContext = SecurityContext.getInstance();

    // Test with ADMIN role
    Agent agent1 = new Agent( id: 1L,  name: "achraf",  password: "1234",  role: "USER");
    Agent agent2 = new Agent( id: 2L,  name: "rachid",  password: "1234",  role: "ADMIN");
    securityContext.login( username: "achraf",  password: "1234");
    Transaction transaction = Transaction.builder()
            .id("01")
            .date(LocalDate.now())
            .montant(1000.0)
            .transactionType(VENTE)
            .build();
    // ca va pas passer car agent1 est un USER
    //agent1.addTransaction(transaction);
    securityContext.logout();
    securityContext.login( username: "aaa",  password: "453");
    securityContext.login( username: "rachid",  password: "1234");
    // ca va passer car agent2 est un admin
    agent2.addTransaction(transaction);
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Users\Achraf HAMMI\AppData\Local\Programs\IntelliJ IDEA Ultimate\
=== Testing Security Aspect ===
Exception in thread "main" java.lang.SecurityException Create breakpoint : User does not have required roles. Required: [ADMIN]
    at achraf.entities.Agent.addTransaction_aroundBody5$advice(Agent.java:23)
    at achraf.entities.Agent.addTransaction(Agent.java:1)
    at achraf.App.testSecurityAspect(App.java:166)
    at achraf.App.main(App.java:36)
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Users\Achraf HAMMI\AppData\Local\
=== Testing Security Aspect ===
user n'existe pas!!!!
NOTIFICATION! rachid a fait la transaction Transaction 01 [2024-12-13, 1000.0, VENTE]
```

FIN DU RAPPORT.
MERCI.