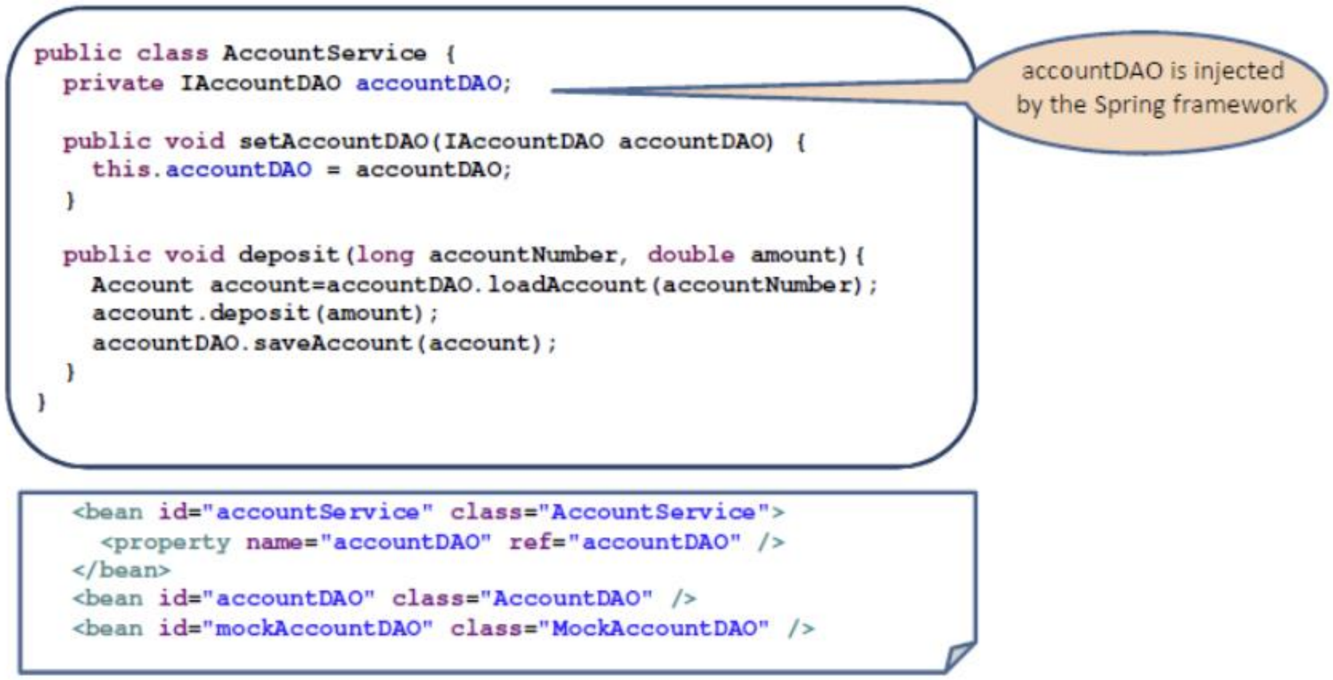
Question 1 [20 points] {20 minutes}

a. Explain what Dependency Injection is.
b. What is the advantage of dependency injection?
c. Give an example of how we implement dependency injection in the Spring framework. Write code snippets that clearly show how we implement dependency injection in the Spring framework.

a. Flexible way to wire objects together.
b. Flexibility. Easy to change the wiring of objects
c.



```java
public class AccountService {
  private IAccountDAO accountDAO;

  public void setAccountDAO(IAccountDAO accountDAO) {
    this.accountDAO = accountDAO;
  }

  public void deposit(long accountNumber, double amount){
    Account account=accountDAO.loadAccount(accountNumber);
    account.deposit(amount);
    accountDAO.saveAccount(account);
  }
}
```

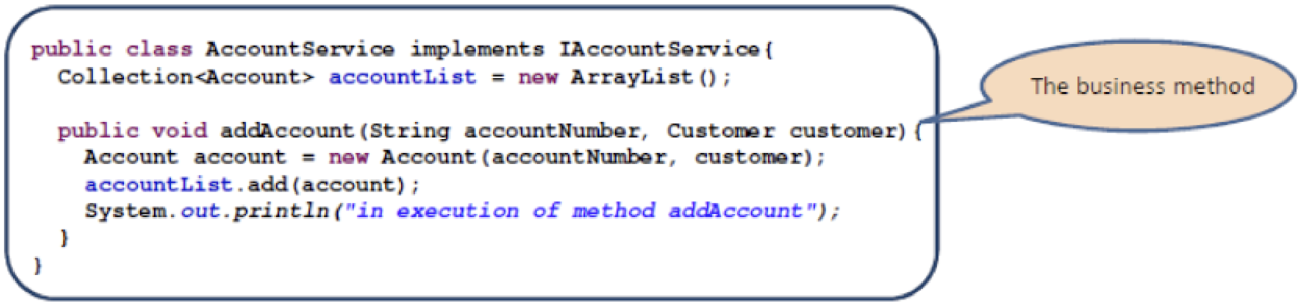
```xml
<bean id="accountService" class="AccountService">
  <property name="accountDAO" ref="accountDAO" />
</bean>
<bean id="accountDAO" class="AccountDAO" />
<bean id="mockAccountDAO" class="MockAccountDAO" />
```

Question 2 [20 points] {20 minutes}

a. Explain what AOP is.
b. What is/are the advantage(s) of AOP?
c. Give an example of how we implement AOP in the Spring framework. Write code snippets that clearly show how we implement AOP in the Spring framework.

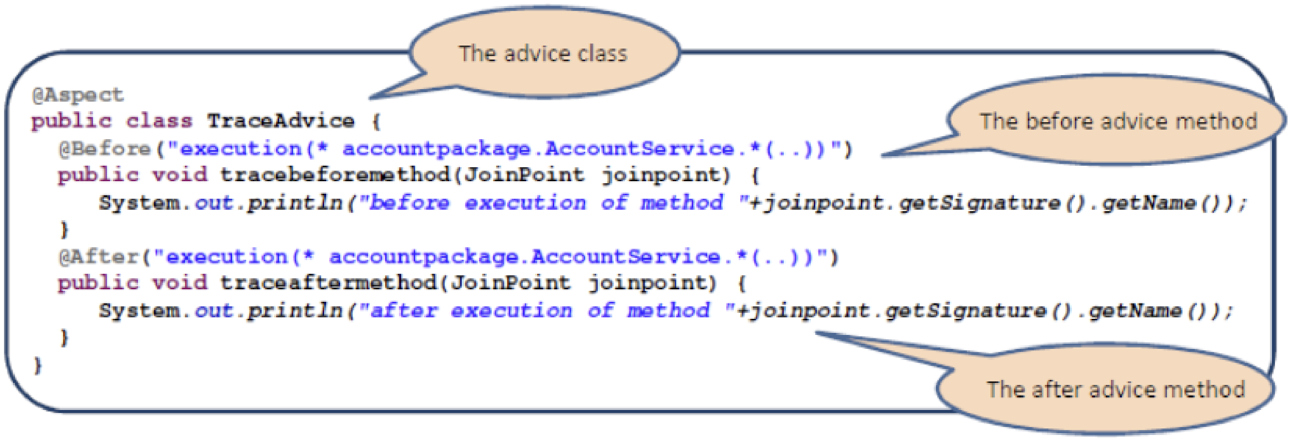
a. Write crosscutting concerns at one place and use them at different places in your application at runtime
b. SOC: Separate functionality at design time, but weave them together at runtime

   DRY: Don't repeat yourself.



```
public class AccountService implements IAccountService{
  Collection<Account> accountList = new ArrayList();

  public void addAccount(String accountNumber, Customer customer){
    Account account = new Account(accountNumber, customer);
    accountList.add(account);
    System.out.println("in execution of method addAccount");
  }
}
```



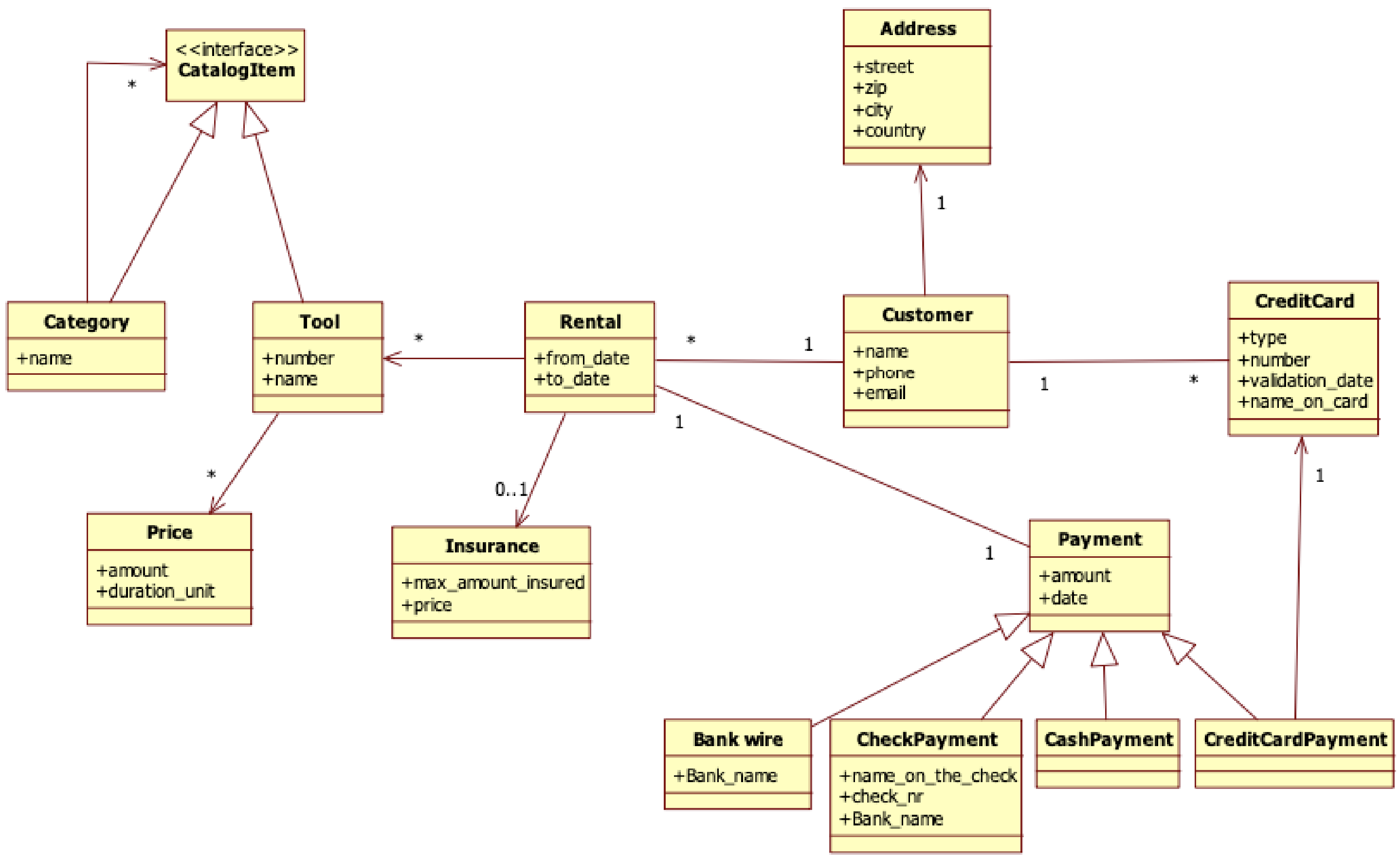
```
@Aspect
public class TraceAdvice {
  @Before("execution(* accountpackage.AccountService.*(..))")
  public void tracebeforemethod(JoinPoint joinpoint) {
     System.out.println("before execution of method "+joinpoint.getSignature().getName());
  }
  @After("execution(* accountpackage.AccountService.*(..))")
  public void traceaftermethod(JoinPoint joinpoint) {
     System.out.println("after execution of method "+joinpoint.getSignature().getName());
  }
}
```

Question 3 [15 points] {25 minutes}

A customer wants us to design a tool rental application with the following requirements:

1. The application keeps track of which customers (name, address, country, phone, email) rent which tools (toolnumber, toolname), price_per_day, price_per_3days, price_per_week, price_per_month,etc) on which days.
2. Every tool has different pricing. For example a certain drill costs $10 per day, but if you rent it 3 days, it costs $20. If you rent it the whole week, it costs $30. For the whole month it costs $50. A certain saw might cost $30 if you rent it 4 days.
3. A customer can rent out multiple tools in one rental.
4. The application keeps track of payments(amount, date)
5. Rentals can be paid with credit card, cash, check or with bank wire. If you pay by check, the application should store the name on the check, the check number and the bank name on the check. If you pay with bank wire, the application should store the name of the bank.
6. Tools are categorized in categories (drilling tools, sawing tools, etc), and within these categories we have subcategories of the different brands. These brands might have other subcategories.
7. Tool rentals can be insured but insurance is not required. If a tool rental is insured, then the application should keep track of the maximum amount that is insured and the price that has to be paid for the insurance.

Draw the class diagram of your design. Make sure you add all necessary UML elements (attributes, multiplicity, etc.) to communicate the important parts of your design.


<<interface>>
CatalogItem
*
Category
+name
Tool
+number
+name
*
Price
+amount
+duration_unit
*
Rental
+from_date
+to_date
0..1
Insurance
+max_amount_insured
+price
*
1
1
Address
+street
+zip
+city
+country
1
Customer
+name
+phone
+email
1
*
CreditCard
+type
+number
+validation_date
+name_on_card
1
1
Payment
+amount
+date
Bank wire
+Bank_name
CheckPayment
+name_on_the_check
+check_nr
+Bank_name
CashPayment
CreditCardPayment

Question 4 [20 points] {20 minutes}

Another customer wants us to design a car rental application with the following requirements:

1. The application keeps track of which customers (name, address) rents which car (licenceNumber, brand, type, price_per_day) on which days.
2. The application keeps track of the list of credit cards that a customer may have
3. The application keeps track of payments(amount, date)
4. Car rentals can only be paid with credit card.
5. Cars are categorized in categories (economy, business(standard, full size, specialty), minivan, suv, etc)

We decide to make a generic rental framework.

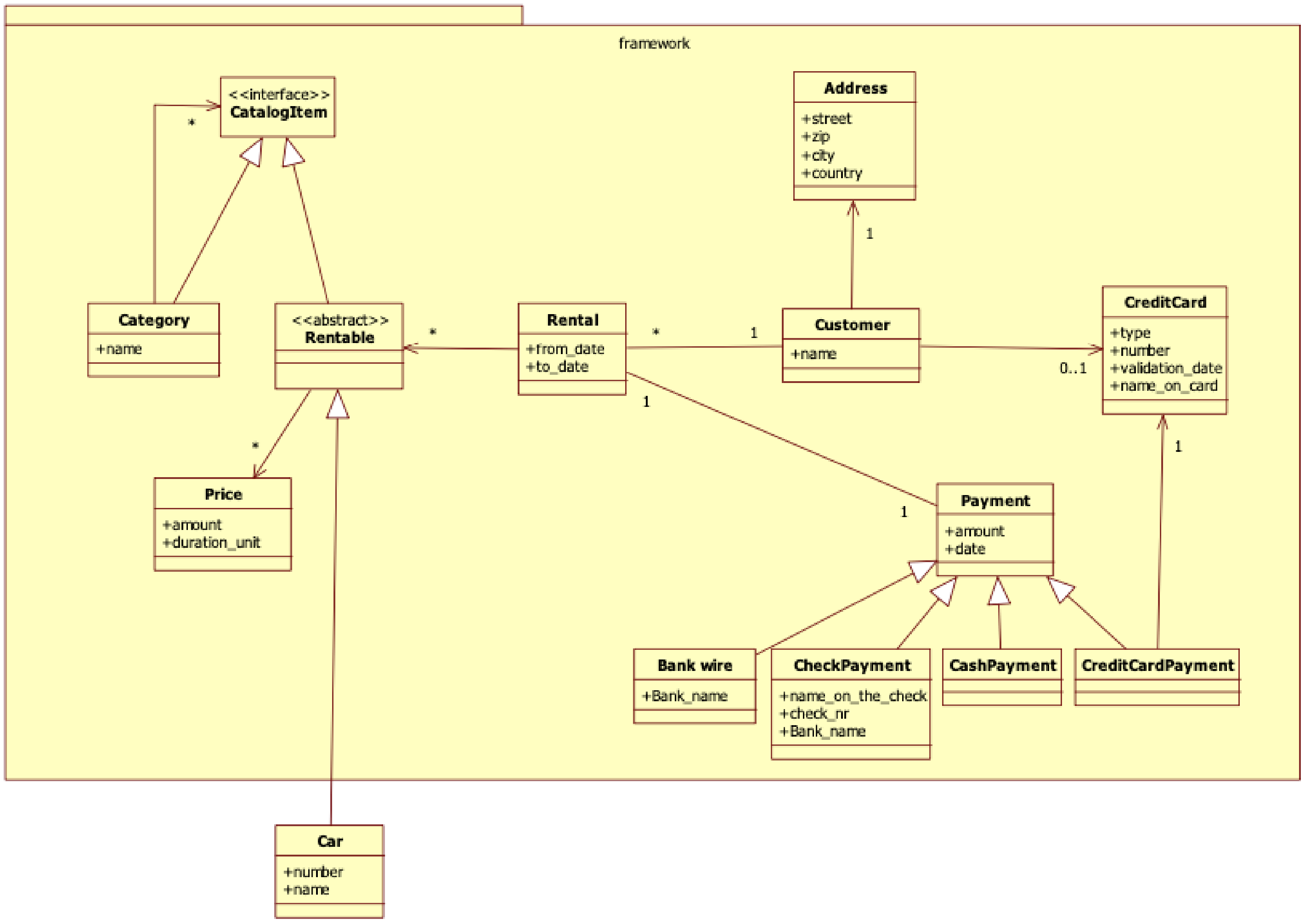
The framework should support the following requirements:

1. The framework keeps track of which customers (name, address) rents which rental-products on which days.
2. A customer can rent out multiple rental-products in one rental.
3. The application keeps track of payments (amount, date)
4. Rentals can be paid with different payment options
5. Rental-products are categorized in categories and subcategories
6. The framework supports all possible ways to specify the rental price: price per hour, price per half day, price per day, price per week, price per month, price per year, price per 2 weeks, etc.

Draw the **class diagram** the design of the car rental application using the framework.

So this class diagram should show the design of the framework, and the design of the car rental application. In the class diagram, show clearly which classes are within the framework, and which classes are outside the framework.

Make sure you add all necessary UML elements (attributes, multiplicity, etc.) to communicate the important parts of your design.

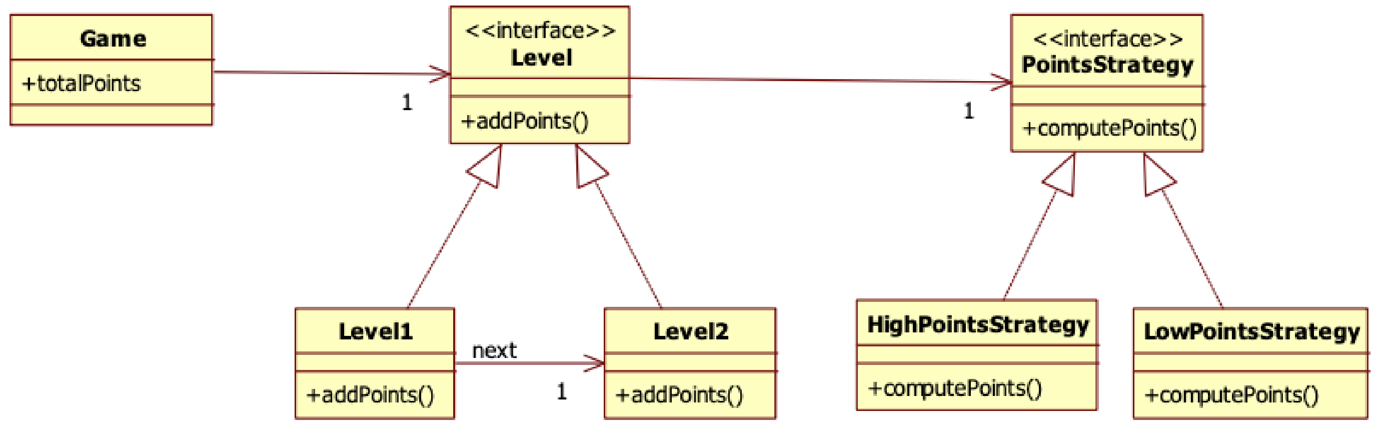

framework
<<interface>>
CatalogItem
*
Address
+street
+zip
+city
+country
1
Category
+name
<<abstract>>
Rentable
*
Rental
+from_date
+to_date
*
1
Customer
+name
CreditCard
+type
+number
+validation_date
+name_on_card
0..1
1
*
Price
+amount
+duration_unit
1
1
Payment
+amount
+date
Bank wire
+Bank_name
CheckPayment
+name_on_the_check
+check_nr
+Bank_name
CashPayment
CreditCardPayment
Car
+number
+name

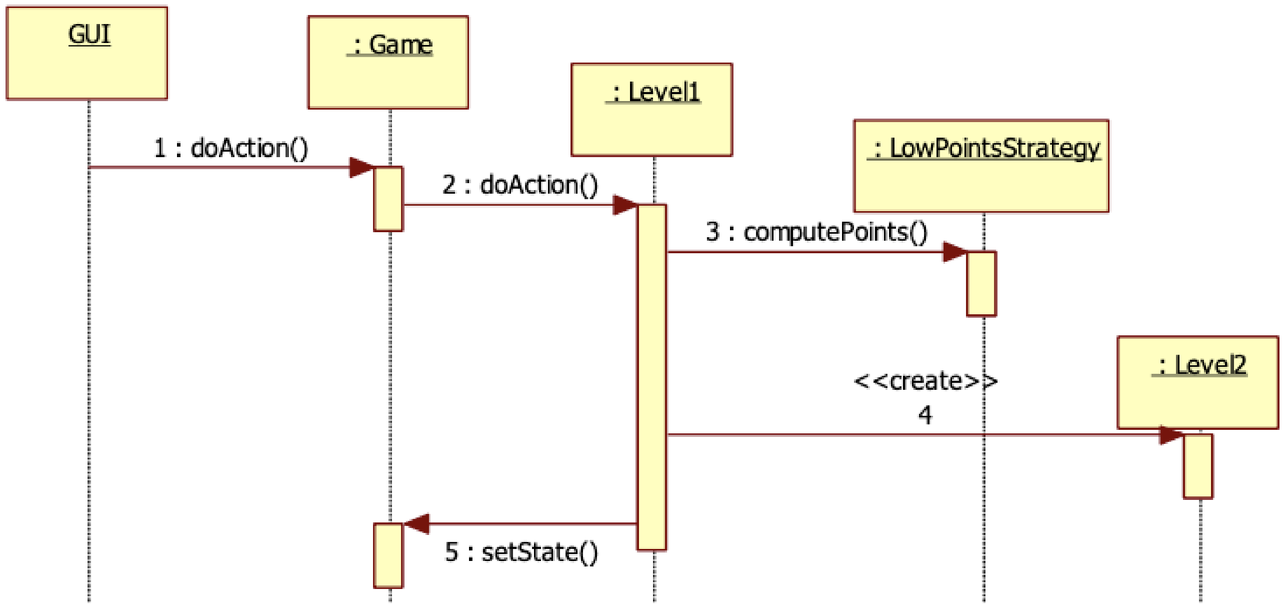
Question 5 [20 points] {25 minutes}

Suppose we need to design a Game which contains multiple levels. You start in level1, and when you collected enough points, you will move up to level2. For some levels, we use the same algorithm to compute the points you can win, but other levels use another algorithm to compute the points you can win. It should be very easy to add new levels to the game.

a. Draw a simple UML class diagram of your design for this game.
b. In a UML sequence diagram show how you advance from one level to another level.

Make sure you add all necessary UML elements to communicate the important parts of your design.


Game
+totalPoints
1
<<interface>>
Level
+addPoints()
1
<<interface>>
PointsStrategy
+computePoints()
Level1
+addPoints()
next
1
Level2
+addPoints()
HighPointsStrategy
+computePoints()
LowPointsStrategy
+computePoints()


GUI
: Game
: Level1
: LowPointsStrategy
: Level2
1 : doAction()
2 : doAction()
3 : computePoints()
<<create>>
4
5 : setState()