CÂU HỎI BỔ SUNG

Cho PROJECT

1. Trình bày hiểu biết của mình về các Biểu đồ UML và ý nghĩa sử dụng của các Biểu đồ UML trong quá trình phát triển phần mềm:
   https://www.uml-diagrams.org/uml-25-diagrams.html
   https://guides.visual-paradigm.com/navigating-the-software-development-journey-a-case-study-of-online-shopping-system-design-with-uml-diagrams/
2. Xây dựng activity diagram/swimlane để thể hiện các hoạt động của Hệ shopping online:
   https://www.visual-paradigm.com/learning/handbooks/software-design-handbook/activity-diagram.jsp
3. Tạo Cart trong một Hệ shopping online là quá trình gồm nhiều bước (search/browser→ add to Cart….) và kết thúc là checkout. Nó được thể hiện trong biểu đồ hoạt động sau (activity diagram). Sinh viên vẽ biểu đồ hoạt động tạo Cart trình bày trong link và giải thích theo hiểu biết của mình: https://www.uml-diagrams.org/online-shopping-uml-activity-diagram-example.html
4. Sinh viên sử dụng link sau để vẽ lại, them, update…Biểu đồ use case của Hệ Shopping online:
   https://www.uml-diagrams.org/examples/online-shopping-use-case-diagram-example.html
5. Tham khảo trang link sau để cập nhật Biểu đồ lớp phân tích và thiết kế. Chỉ ra mình đã cập nhật gì:
   https://www.uml-diagrams.org/examples/online-shopping-domain-uml-diagram-example.html
   https://www.uml-diagrams.org/class-diagrams-overview.html
6. Trình bày các quan hệ và code java tương ứng giữa các lớp https://www.sitesbay.com/java/java-relationship-in-java
7. Vẽ biểu đồ lớp thể hiện quan hệ lớp của code java sau đây:

```
import java.util.*;
class Product{
private String name;
private double price;
Product(String name, double price){
this.name = name;
this.price = price;
}
double getPrice(){
return price;
}
}
class Basket {
private List<Product> products;
public Basket() {
products = new ArrayList<>();
```

```
}
public void addProduct(Product product) {
products.add(product);
}
public double getTotal() {
double total = 0;
for (Product product : products) {
total += product.getPrice();
}
return total;
}
}
class Order {
public double calculateTotal(Basket basket) {
return basket.getTotal() + 5.0; // Add 5.0 for shipping
}
}
public class Main {
public static void main(String[] args) {
Basket basket = new Basket();
basket.addProduct(new Product("Item 1", 10.0));
basket.addProduct(new Product("Item 2", 20.0));
Order order = new Order();
double total = order.calculateTotal(basket);
System.out.println("Total: " + total); // Output: Total: 35.0
}
}
```

Navigating the Software Development Journey: A Case Study of Online Shopping System Design with UML Diagrams

Table of Contents hide

1 Introduction

1.1 Understanding the Software Design Process

1.2 The Role of Use Cases in Software Design

1.2.1 Elaborating Use Cases

1.3 Case Study: Online Shopping System

1.3.1 Requirements Gathering:

1.3.2 High-Level Design:

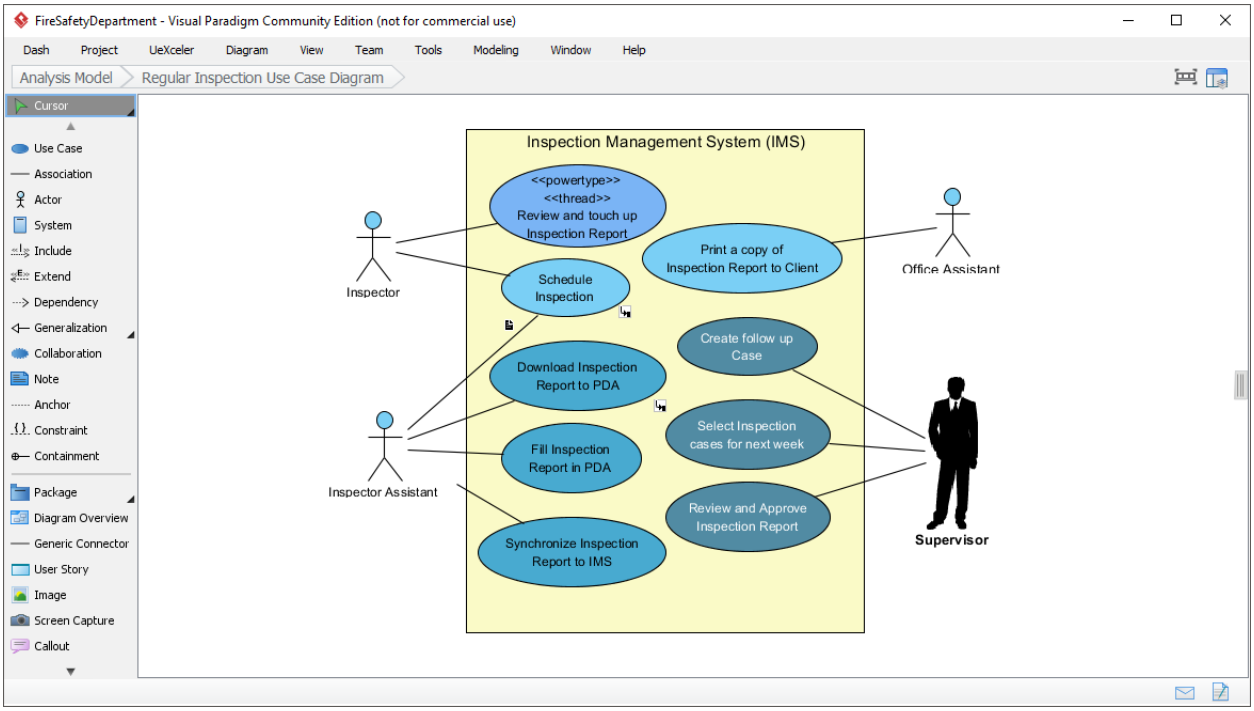
1.3.3 Detailed Design:

1.3.4 Implementation:

1.4 Summary

Introduction

In the realm of software development, the journey from concept to fully functional application is a complex and multifaceted process. A crucial aspect of this journey is the art of translating high-level requirements into detailed, actionable designs. Unified Modeling Language (UML) diagrams serve as invaluable tools for developers and stakeholders alike, enabling a comprehensive understanding of a system's architecture and behavior. In this article, we embark on a guided tour of the software development process by immersing ourselves in a real-world case study: the creation of an Online Shopping System. Through the lens of UML diagrams, we will explore how this intricate system is meticulously designed and executed, from its inception to deployment.



Understanding the Software Design Process

Before diving into the specifics of elaborating use cases with UML diagrams, let's briefly outline the software design process. It typically consists of the following stages:

a. Requirements Gathering: This is the initial phase where stakeholders and developers gather and document high-level requirements for the software.

b. High-Level Design: In this stage, a broad architectural overview of the system is created, outlining major components and their interactions.

c. Detailed Design: Here, the high-level design is broken down into finer details, specifying the behavior of individual components.

d. Implementation: Developers write the actual code based on the detailed design.

e. Testing: The software is tested to ensure it meets the specified requirements.

The Role of Use Cases in Software Design

Use cases are essential for understanding and documenting how a system will interact with its users or external systems. They describe various scenarios or interactions between the system and its users, helping to define the system's functionality.

Elaborating Use Cases

To elaborate use cases effectively, we'll discuss how to use UML diagrams in different development stages.

a. High-Level Use Case Diagrams:

- **Use Case Diagrams**: In the high-level design phase, start with use case diagrams. These diagrams provide an overview of the system's major use cases and their relationships. They help stakeholders understand the system's main functionalities.

- **Actor-Use Case Mapping**: Identify the actors (users, external systems, etc.) and associate them with relevant use cases. This clarifies who interacts with the system and how.

b. Detailed Use Case Diagrams:

- **Activity Diagrams**: As you move to the detailed design phase, create activity diagrams for each use case. These diagrams illustrate the flow of activities within a use case, including conditional and parallel paths. They help developers understand the sequence of actions required to achieve specific goals.
- **Sequence Diagrams**: Sequence diagrams show the interactions between objects or components in a use case. They are particularly useful for detailing the dynamic behavior of the system, showing how objects collaborate to accomplish tasks.
- **State Diagrams**: For use cases involving complex states or state transitions, state diagrams can be valuable. They depict how an object or system transitions between different states in response to events.

c. Implementation and Testing:

- **Class Diagrams**: During implementation, class diagrams play a crucial role. They define the structure of classes, their attributes, and relationships, facilitating the coding process.
- **Component Diagrams**: Component diagrams help developers visualize the physical arrangement of system components and their dependencies. This aids in ensuring a well-organized and maintainable codebase.
- **Deployment Diagrams**: In the deployment phase, deployment diagrams come into play. They illustrate how software components are distributed across hardware nodes or servers, helping ensure efficient system deployment.

4. Benefits of Elaborating Use Cases with UML Diagrams

    - **Clarity**: UML diagrams provide a visual representation of complex systems, making it easier for stakeholders, developers, and testers to understand and communicate about the system's design.
    - **Consistency**: UML promotes consistency in design by offering a standardized way to document various aspects of a software system.

- **Documentation**: UML diagrams serve as valuable documentation that can be referenced throughout the software development lifecycle.
- **Error Reduction**: By visualizing the system's behavior, UML diagrams help identify design flaws and inconsistencies early in the process, reducing costly errors in later stages.

Case Study: Online Shopping System

Elaborating use cases with UML diagrams is a critical part of the software design process. From high-level requirements to detailed modeling, UML diagrams provide a structured approach to understanding and documenting a system's functionality and behavior. By following these guidelines, software development teams can create robust, well-designed applications that meet the needs of their users and stakeholders.

Let's walk through the software development process for an online shopping system, applying various UML diagrams at different stages to illustrate the development process.

Requirements Gathering:

Imagine a client, XYZ Electronics, wants to develop an online shopping system. The high-level requirements include user registration, product browsing, shopping cart management, order placement, and payment processing.

Use Case Diagram: Create a high-level use case diagram showing actors like "Customer" and "Admin" interacting with use cases like "Browse Products," "Add to Cart," and "Place Order."

High-Level Design:

In this phase, we outline the system's architecture and major components.

Component Diagram: Create a component diagram showing high-level components like "Web Server," "Database Server," and "Payment Gateway." These illustrate the major parts of the system.

Detailed Design:

In this phase, we elaborate on the use cases and components in greater detail.

Activity Diagram (Use Case – Browse Products): Create an activity diagram detailing the steps a customer takes to browse products. It includes actions like "Search Products," "View Product Details," and "Add to Cart."
Sequence Diagram (Use Case – Place Order): Develop a sequence diagram illustrating the interactions between the "Customer," "Shopping Cart," "Order," and "Payment Gateway" during an order placement process.
Class Diagram: Design class diagrams for key entities like "Product," "Customer," "Shopping Cart," "Order," and "Payment." Define attributes, methods, and relationships between these classes.

Implementation:

Developers write code based on the detailed design. This includes building the user interface, backend logic, and database integration.

Deployment Diagram: Create a deployment diagram illustrating how the application components are distributed across servers, such as "Web Server" and "Database Server."

Testing:

Quality assurance teams conduct various tests, including functional, integration, and user acceptance testing, to ensure the system functions as expected.

Now, let's see how these UML diagrams play a role in this case study:

- **Use Case Diagram**: The high-level use case diagram serves as a reference point for stakeholders to understand the system's primary functionalities, such as browsing, shopping, and ordering.
- **Component Diagram**: This diagram helps architects and developers plan the system's structure by identifying major components like servers and databases.
- **Activity Diagrams**: These guide developers in implementing the step-by-step flow for specific actions like browsing products or adding items to the cart.
- **Sequence Diagrams**: Sequence diagrams depict the dynamic interactions between objects and components, aiding developers in writing code that reflects these interactions.
- **Class Diagram**: Class diagrams ensure consistency in the data structures used throughout the system, helping developers implement core entities accurately.

- **Deployment Diagram**: Deployment diagrams assist in deploying the system, ensuring that components are properly distributed across servers or cloud infrastructure.

By applying UML diagrams at each stage of the development process, the online shopping system's design and implementation become more structured and organized. This approach not only helps developers and designers but also provides a clear visual reference for clients and stakeholders, ultimately leading to a successful and efficient software development project.

Summary

In the software development world, success hinges on meticulous planning and execution. This article, "Navigating the Software Development Journey: A Case Study of Online Shopping System Design with UML Diagrams," takes you on a journey through the development of an online shopping system. We dive into the software design process, beginning with requirements gathering and progressing through high-level design, detailed design, implementation, and testing. At each stage, we demonstrate the indispensable role of UML diagrams, showcasing their ability to provide clarity, consistency, and documentation. Whether you're a developer, designer, or stakeholder, this case study illustrates the power of UML in turning conceptual ideas into functional software systems.