Topic 6: object -oriented Design Implementation

❖Programming and Development Process

- Coding is an end goal of software development.
- Iterative and incremental development process results in the feeding of prior iteration into the beginning of next iteration, continuously refining the implementation works.

❖ Mapping Design to code

- The goal of mapping design to code in Object-Oriented Analysis and Design (OOAD) is to transform our concepts and blueprints into functional software.
- We convert our designed ideas—such as classes, objects, and relationships—into the programming language. To do this, we must translate our models and diagrams into computer-readable code.
- It requires writing code for

a) Class and interface definitions
b) Method definitions

➢Class definitions are created by mapping design class diagrams to code.

➢Method definitions are created by mapping interaction diagrams to code.

❖ Importance of Mapping Design to Code

- **Maintainability:**
  - Software maintenance and updates are made simpler when the design is successfully translated to code.
- **Encouraging Collaboration:**
  - By translating design into code, developers, designers, and stakeholders can communicate in a common language.
- **Increasing Development Efficiency:**
  - By offering a precise implementation roadmap, design-to-code mapping expedites the development process.

- **Enforcing Design Principles:**
  - By adhering closely to the design during the coding phase, developers ensure that the software aligns with established design principles and best practices.
- **Improving Debugging and Testing:**
  - It is simpler to debug and test software when design elements are faithfully reflected in the code.

Techniques for Mapping Classes to Code

Mapping classes to code involves translating the design of your classes, including their properties and behaviors, into actual programming code.

1. Identify Classes

- Begin by identifying the classes in your design. Classes represent objects or entities in your system and typically correspond to nouns in your problem domain.

2. Define Properties

- For each class, define its properties or attributes. These are the characteristics that describe the state of the object. Map each property to a corresponding data type in your programming language.

3. Define Methods

- Determine the behaviors or operations that each class can perform. These are represented as methods or functions.

4. Encapsulation

5. Inheritance

6. Composition

7. Interfaces and Abstract Classes

8. Dependency injection

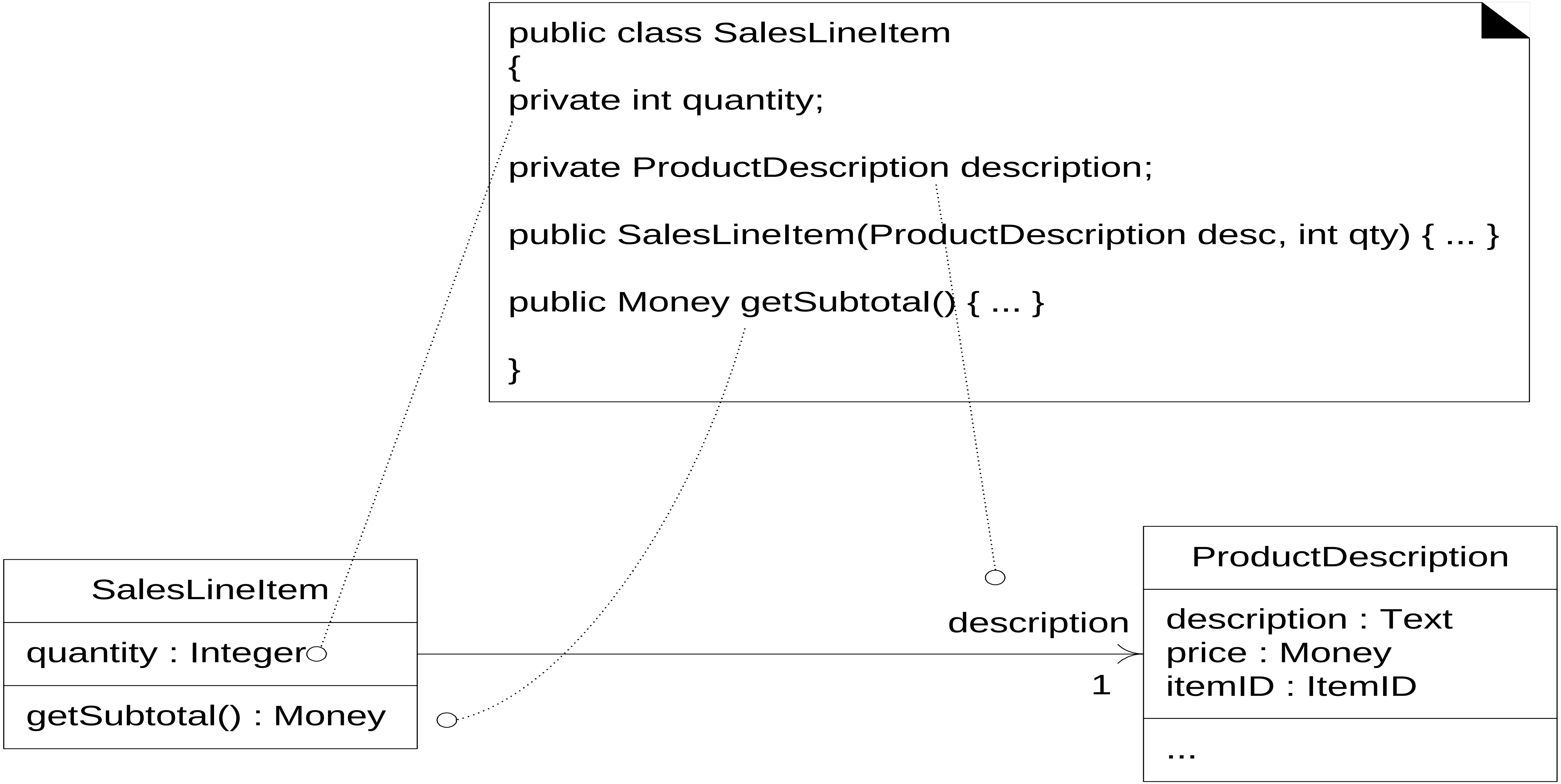
- When classes depend on each other, use dependency injection techniques to provide the required dependencies. This promotes loose coupling and facilitates testing and maintenance.

9. Design patterns

10. Coding Standards and Conventions

11. Testing

Creating Class Definitions from Design Class Diagram



```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

public SalesLineItem(ProductDescription desc, int qty) { ... }

public Money getSubtotal() { ... }

}
```

SalesLineItem

quantity : Integer

getSubtotal() : Money

ProductDescription

description : Text
price : Money
itemID : ItemID

...

Creating Class Definitions from Design Class Diagram

- Class diagrams visually represent the structure and relationships between objects, including their attributes (data) and methods (behaviors). Once you have a class diagram, you can translate it into class definitions in your programming language of choice (e.g., Python, Java, C++).

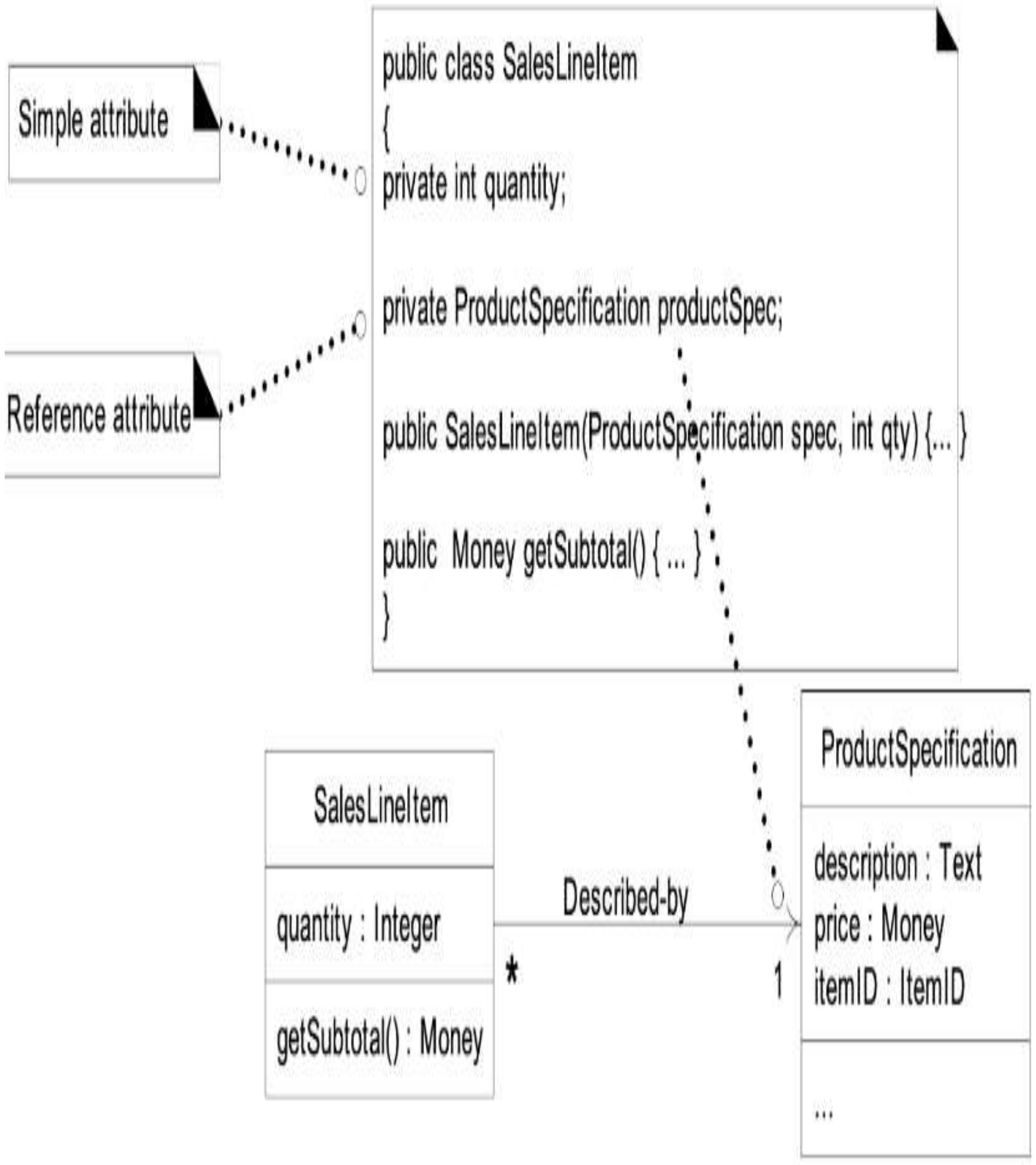
When creating class definitions from a design class diagram:

- Identify the **classes** from the diagram and translate them into class definitions.
- Define **attributes** and **methods** based on the diagram.
- Implement relationships between classes, such as inheritance, composition, or aggregation.
- Ensure that the interactions and behaviors in the class diagram are reflected in the code through method calls and object references.

Adding Reference Attributes

- A reference attribute is an attribute that refers to another complex object, not to a primitive type such as a String, Number, and so on.

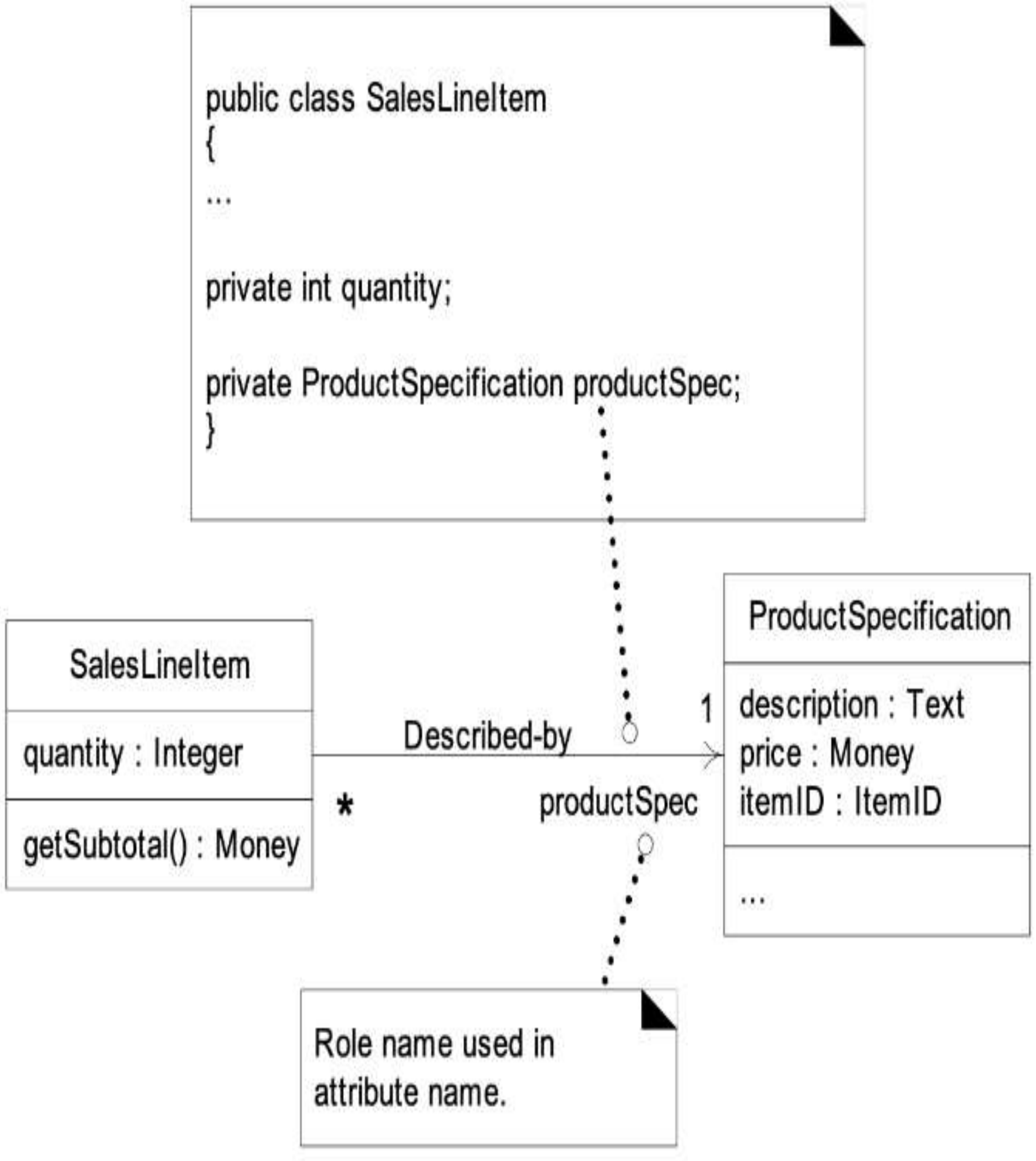
- The reference attributes of a class are suggested by the associations, aggregation and composition and navigability in a class diagram.



```
public class SalesLineItem
{
private int quantity;

private ProductSpecification productSpec;

public SalesLineItem(ProductSpecification spec, int qty) {... }

public  Money getSubtotal() { ... }
}
```

Adding roles names

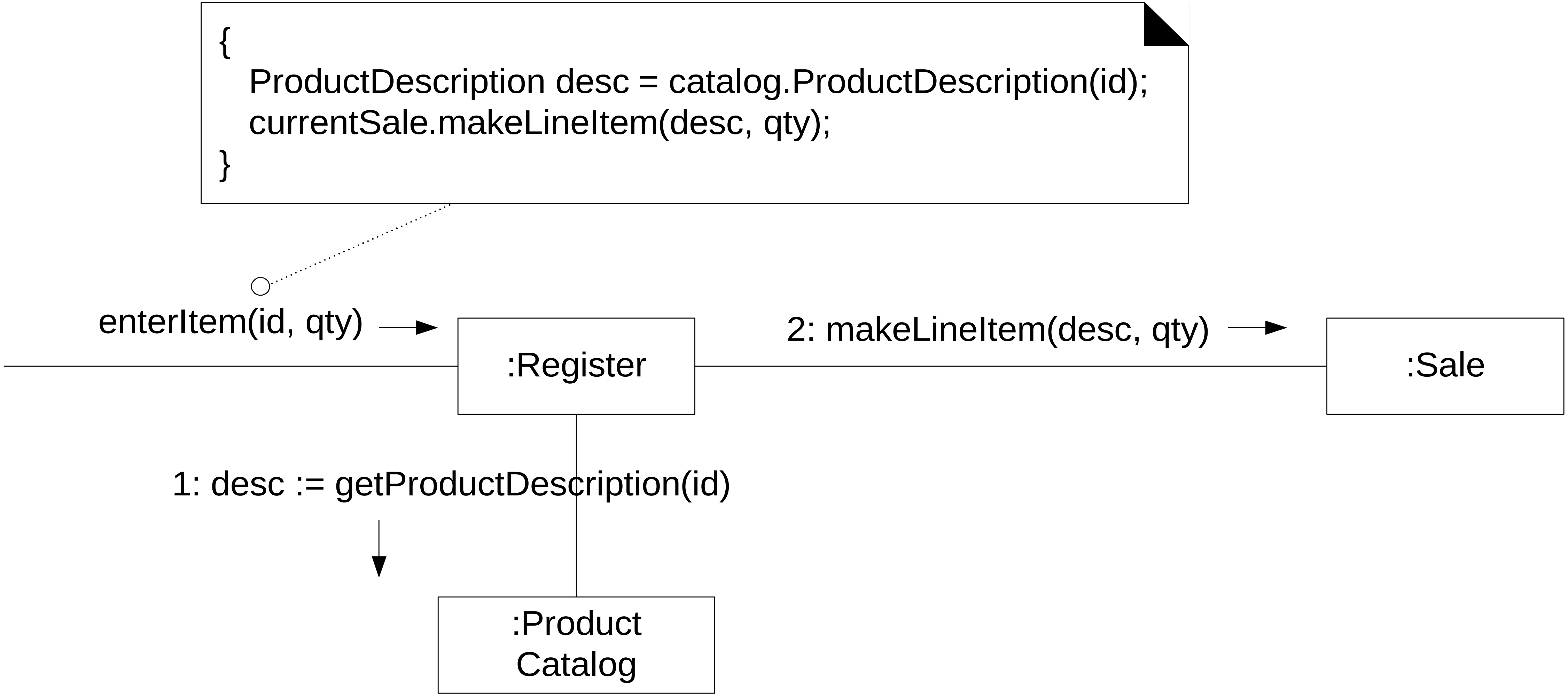
- The next iteration will explore the concept of role names in static structure diagrams. Each end of an association is called a role. Briefly, a role name is a name that identifies the role and often provides some semantic context as to the nature of the role.
- If a role name is present in a class diagram, use it as the basis for the name of the reference attribute during code generation.



```
public class SalesLineItem
{
...

private int quantity;

private ProductSpecification productSpec;
}
```

❖ Creating methods from collaboration Diagram

- The sequence diagram consists of sequence of messages which are translated to a series of statements in the method definitions.



```
{
   ProductDescription desc = catalog.ProductDescription(id);
   currentSale.makeLineItem(desc, qty);
}
```

enterItem(id, qty)
:Register
2: makeLineItem(desc, qty)
:Sale
1: desc := getProductDescription(id)
:Product Catalog

❖ Updating Class Definitions

- One to many relationships are common.
- Such relationships is implemented using collection object such as list, map or array.
- The choice of collection class is influenced by the requirements. i.e. key based lookup requires Map while growing ordered list requires a List.
- If object implements an interface, declare the variable in terms of the interface.

❖ Exception and error Handling

Exception

- An exception is a condition that is caused by a runtime error in the program.
- An exception may occur due to following reasons:

a) Invalid data entered by a user.

b) File to be opened can not be found.

c) The network connection has lost in the middle of the communication

Sources for Exceptions

1. User errors
2. Programmer errors
3. Physical resource failure

Categories of Exception

1. Checked Exception

- It is the exception that can not be foreseen by the programmer.
- Eg: FileNotFoundException

2. Runtime Exception

- It is the exception that could be avoided by the programmer.
- It is ignored at the time of compilation.

3. Errors

- They are the problems beyond the control of user and programmer.
- Eg: StackOverflowException

Exception handling process

- In object-oriented programming languages, there is a mechanism to handle exceptions in a proper manner.
- Try, throw and catch are the basic exception handling paradigms used.
- The general code is put in try block. It means try to execute the code.
- If the system succeeds to execute the code, execution flows in general or normal order.
- If something goes wrong while executing the try block, this code throws an exception object and stops executing code of try block.
- The error handler catches the exception object and make necessary actions needed.
- Execution continues with the next instructions following the catch block.

```
try:

    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))
    result = numerator / denominator
except ZeroDivisionError:

    print("Error: Division by zero is not allowed.")
except ValueError:

    print("Error: Invalid input. Please enter numeric values.")
else:

    print(f"The result is: {result}")

finally:

    print("Execution completed.")
```

THANK YOU