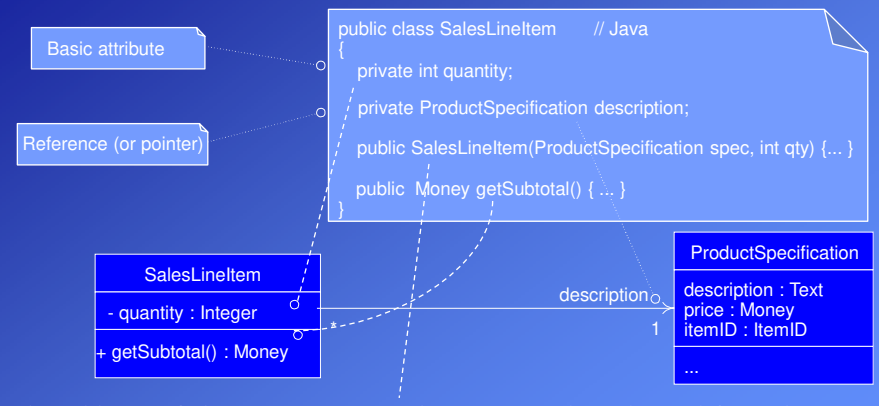
Coding (Implementation)

The design model (design class diagram and interaction diagrams) provides some of the information that is necessary to generate a part of the code.

Creating Class Definitions from Design Class Diagrams:

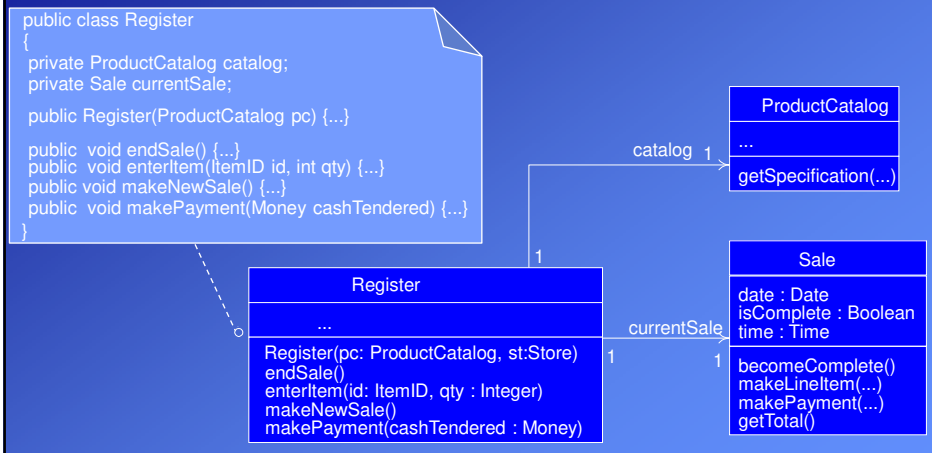


The addition of the constructor to the source code is derived from the create(spec, qty) message sent to a SalesLineItem in the enterItem interaction diagram (see 5.6).

Creating Methods from Interaction Diagrams:

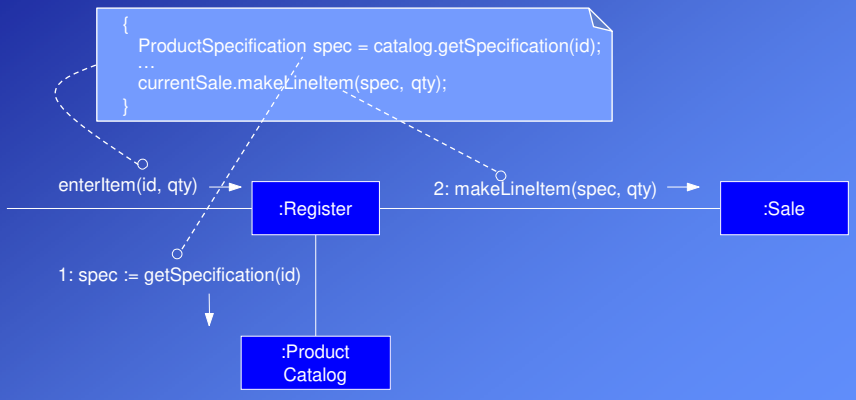
The sequence of the messages in an interaction diagram is mapped to statements in the method definitions.

Example: The Register class and the enterItem method



Example: The Register class and the enterItem method (cont'd)

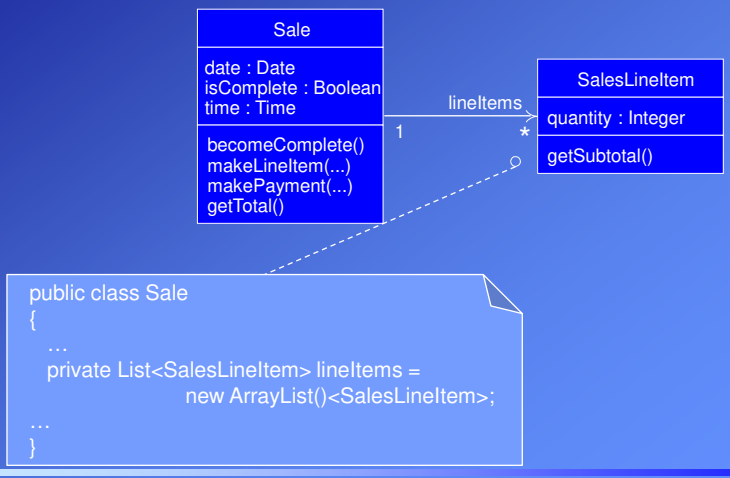
Messages numbered as 1 and 2 are mapped to statements in the enterItem method of the Register class.



Collection Classes in Code:

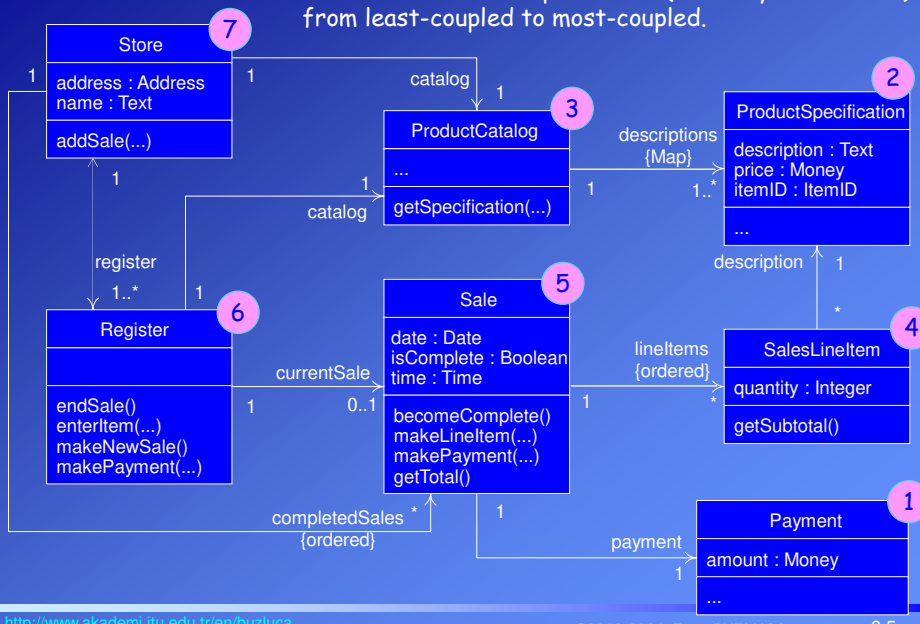
One-to-many relations implemented with the introduction of a **collection object**, such as a List, Map, or even a simple array.

Example: SaleLineItems in the Sale.



Order of Implementation

Classes need to be implemented (and fully unit tested) from least-coupled to most-coupled.



Test-Driven (Test-First) Development

Unit testing: In object oriented programming classes are main individual components (units) of the program.

So classes can (and must) be tested as individual units separately.

In test-driven (or test-first) development, unit testing code is written **before** the code of the class to be tested,

and the developer writes unit testing code nearly for all production code.

The testing code should perform following operations:

- Creating (and deleting) objects of the class (testing the constructor and destructor).
- Sending messages to objects and checking the results.
- Testing exceptional cases by sending parameters out of acceptable range.

All of the testing code is not written once.

The basic rhythm is to write a little test code, then write a little production (class) code, make it pass the test, then write some more test code, and so forth.

Popular unit testing frameworks:
Junit: Java, http://www.junit.org
Nunit: .NET, http://www.nunit.org

CruiseControl: Open Source
http://cruisecontrol.sourceforge.net/

Advantages of test-first programming:

- **It ensures that the unit tests are really written.**
  Otherwise, programmers usually skip unit tests and leave them as the last work.
- **Motivation (satisfaction) of programmers**
  In the traditional style (*test-last development*),
  A developer first writes the code of the class,
  Informally debugs it, and then, as an afterthought, is expected to add unit tests.
  It doesn't feel satisfying.
  However, if the test is written first, we feel a challenge and question in front of us: "Can I write code to pass this test?"
  After the code passes the tests, there is some feeling of success.
  It's human psychology.

Advantages of test-first programming (contd):

- **Clarification of detailed interface and behavior of the class**
  As you write the test code first, you must imagine that the object code exists.
  You must think through the details of the public view of the method: its name, return value, parameters, and behavior.
  That improves the quality of the code.
- **The confidence to change things**
  When a developer needs to change existing classes, a unit test can be run, providing immediate feedback if the change causes an error.

Example: Unit test of the Sale class

Remember that we do not write all the unit tests for Sale first; rather, we write only one test method, implement the solution in class Sale to make it pass, and then repeat.

Assume that we use the unit testing framework Junit.

To use JUnit, you must create a test class that extends the JUnit TestCase class; your test class inherits various unit testing behaviors.

Test code for the makeLineItem method of the Sale class

```
public class SaleTest extends TestCase
{
   public void testMakeLineItem(){
     Sale fixture = new Sale();            // Tested object
     Money total = new Money(7.5);    // Supporting objects
     Money price = new Money(2.5);
     ItemID id = new ItemID(1);
     ProductSpecification spec = new ProductSpecification(id, price, "product 1"); //Supporting
        // Method is being tested ...
     fixture.makeLineItem(spec, 1);
     fixture.makeLineItem(spec, 2);
     assertTrue( fixture.getTotal().equals(total));   // Is the total calculated correctly?
     :::
```


Exceptions and Error Handling

To focus on the responsibility assignment and object design, we have ignored exception handling so far in coding.

However, in application development, exception handling strategies should be considered during design modeling and during implementation (coding).

Coding Examples

The code for the NextGen POS is generated from the design class diagrams and interaction diagrams defined in the design work.

The given example codes define simple cases; they are not, fully developed programs with exception handling, and so on.

Same programs are written in Java and C++.

Please refer to following files.

Java: NextGenPos_java.pdf

C++: NextGenPos_cplusplus.pdf