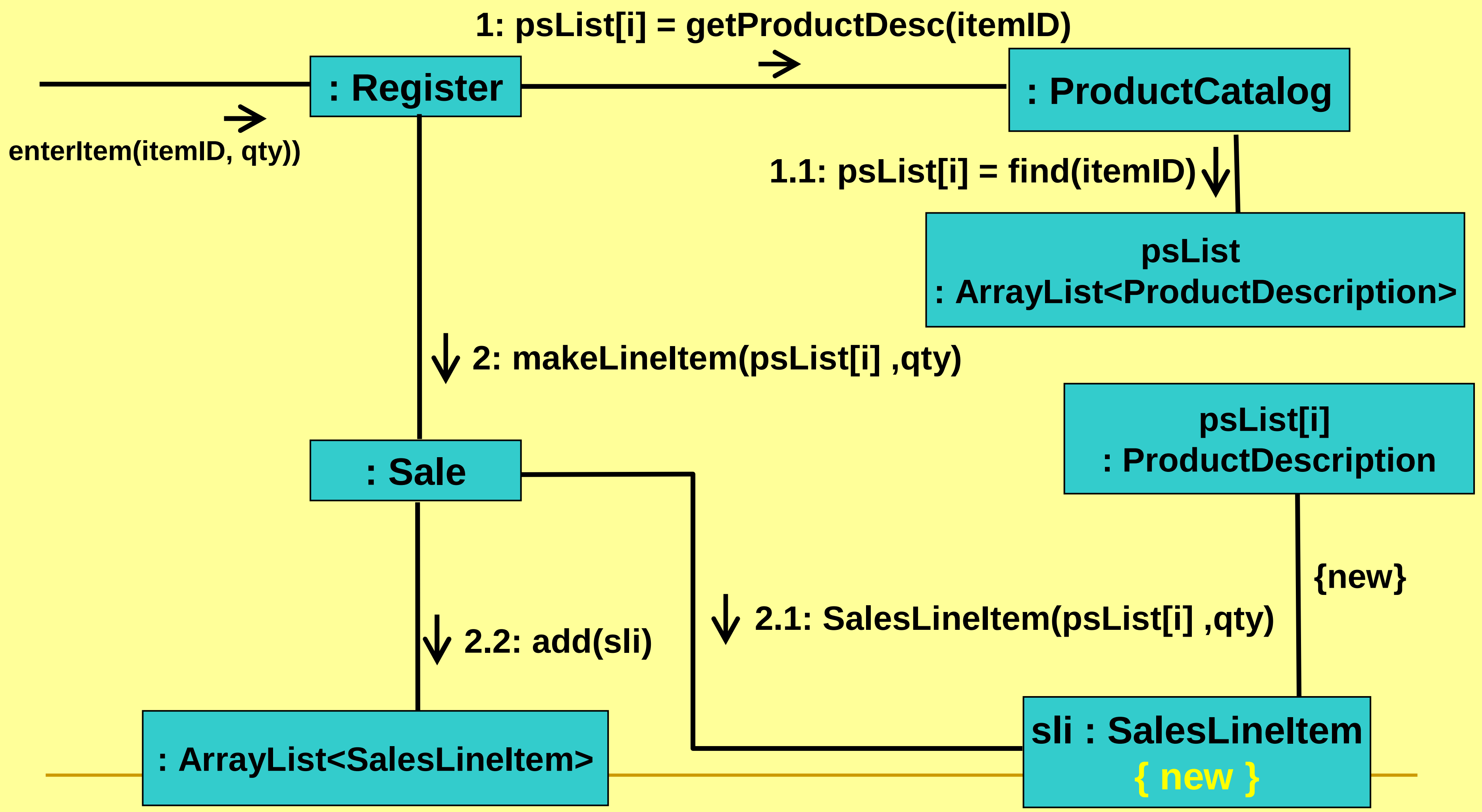
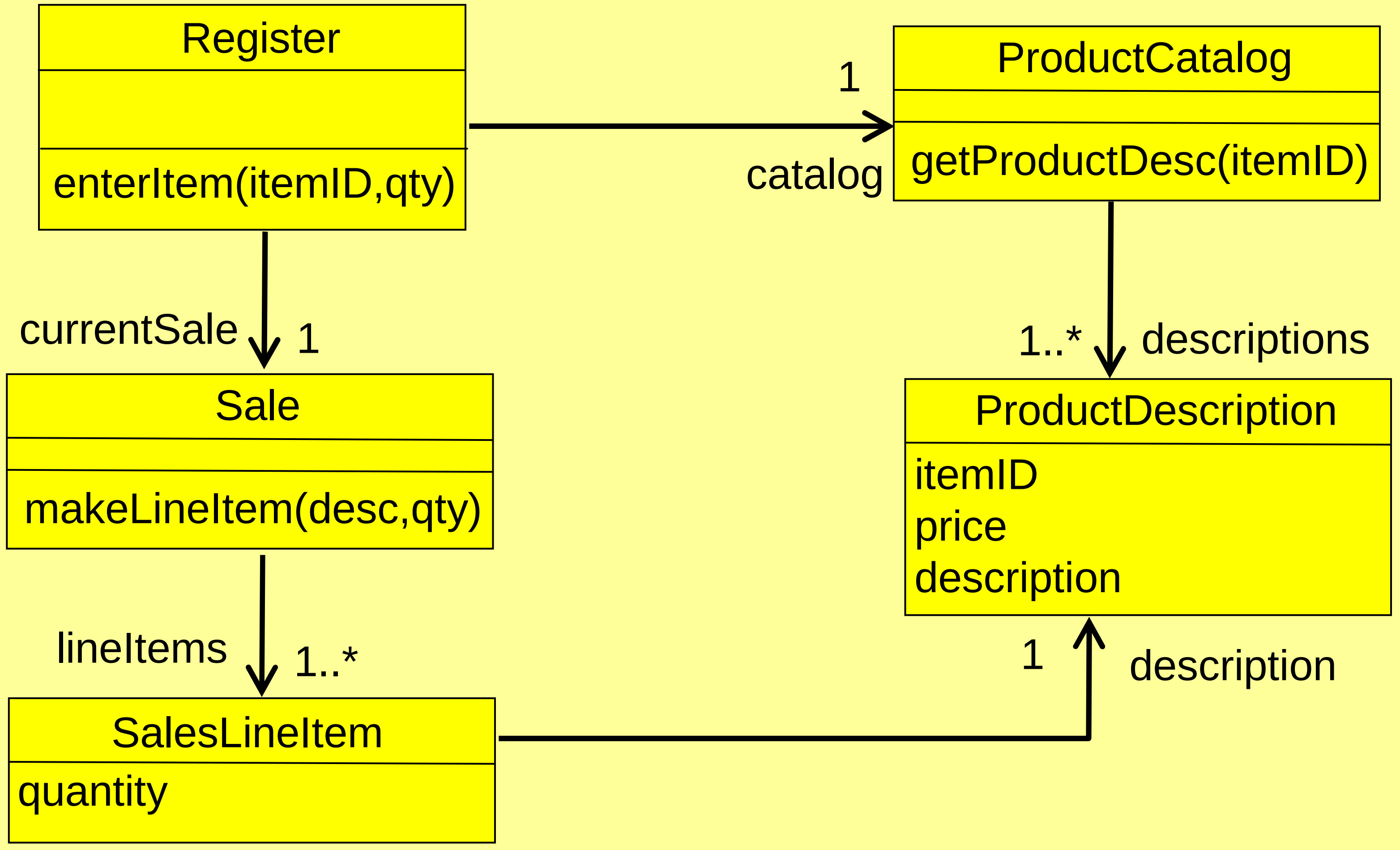
Mapping Designs to Code

INTRODUCTION

- The interaction diagrams and design class diagrams created during design provide some of the necessary input for generating code.
- In this chapter, we will see how to map those artifacts to code in an object-oriented language.

- The following interaction and class diagrams will be used to show the mapping process:


1: psList[i] = getProductDesc(itemID)
: Register
: ProductCatalog
enterItem(itemID, qty))
1.1: psList[i] = find(itemID)
psList
: ArrayList<ProductDescription>
2: makeLineItem(psList[i] ,qty)
psList[i]
: ProductDescription
: Sale
{new}
2.2: add(sli)
2.1: SalesLineItem(psList[i] ,qty)
: ArrayList<SalesLineItem>
sli : SalesLineItem
{ new }

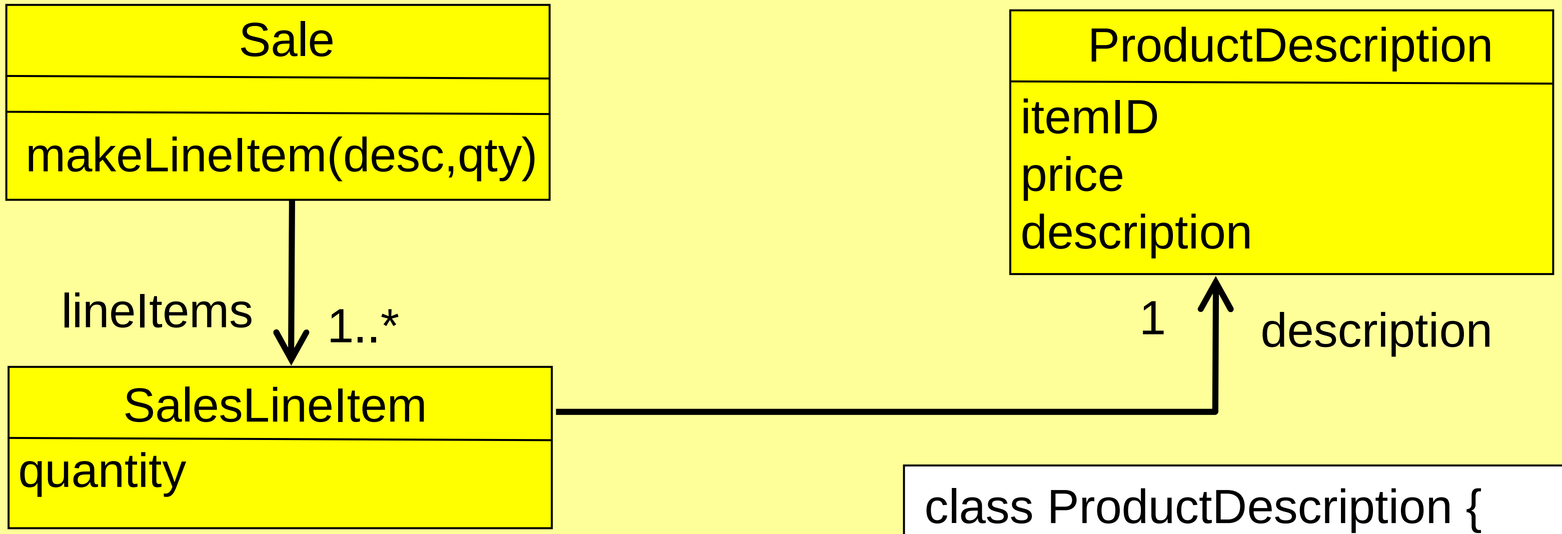

Register
enterItem(itemID,qty)
1
catalog
ProductCatalog
getProductDesc(itemID)
currentSale
1
Sale
makeLineItem(desc,qty)
1..*
descriptions
ProductDescription
itemID
price
description
lineItems
1..*
SalesLineItem
quantity
1
description

CREATING CLASS DEFINITIONS FROM DESIGN CLASS DIAGRAMS

- Basic class definitions can be written from the design class diagrams. The following information can be extracted:
  - Class name
  - Attributes: name, type and access specifier
  - Method: name, return type, parameters and their types, and its access specifier

- Example:

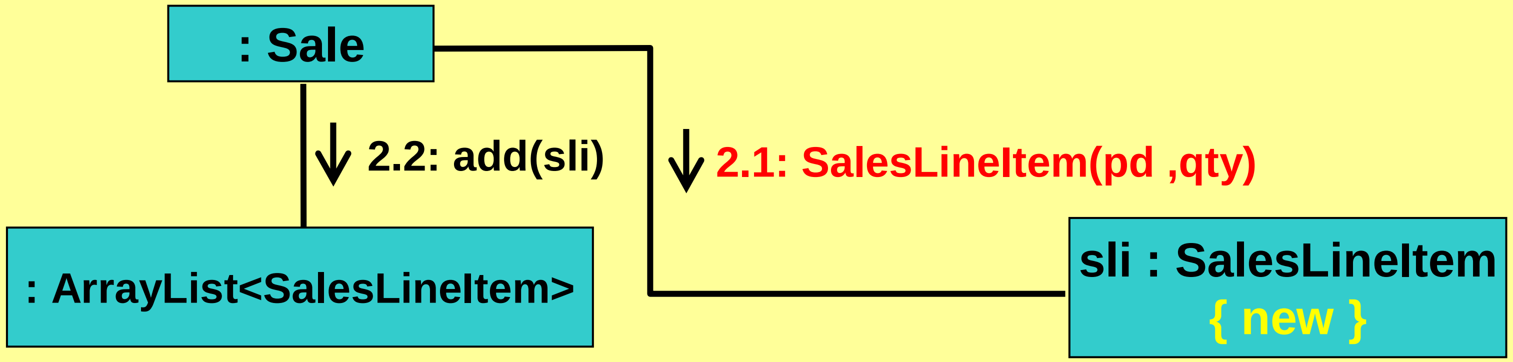
```
class Sale {
        private Vector lineItems;
        public void makeLineItem(ProductDescription desc,
                                 int qty) { }
}
```



```
class ProductDescription {
        private ItemID itemID;
        private Money price;
        private String description;
}
```

```
class SalesLineItem {
        private int quantity;
        private ProductDescription description;
        public SalesLineItem(ProductDescription pd, int qty) { }
}
```

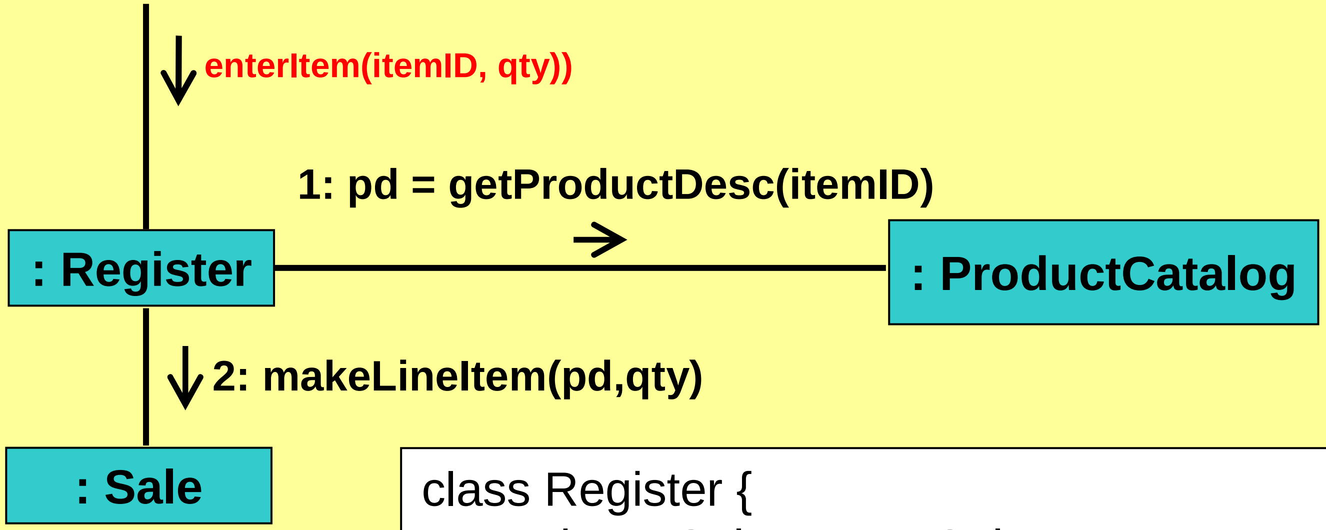
- Note that the constructor in the class **`SalesLineItem`** is derived from the creation of the **`SalesLineItem`** object in the interaction diagram.



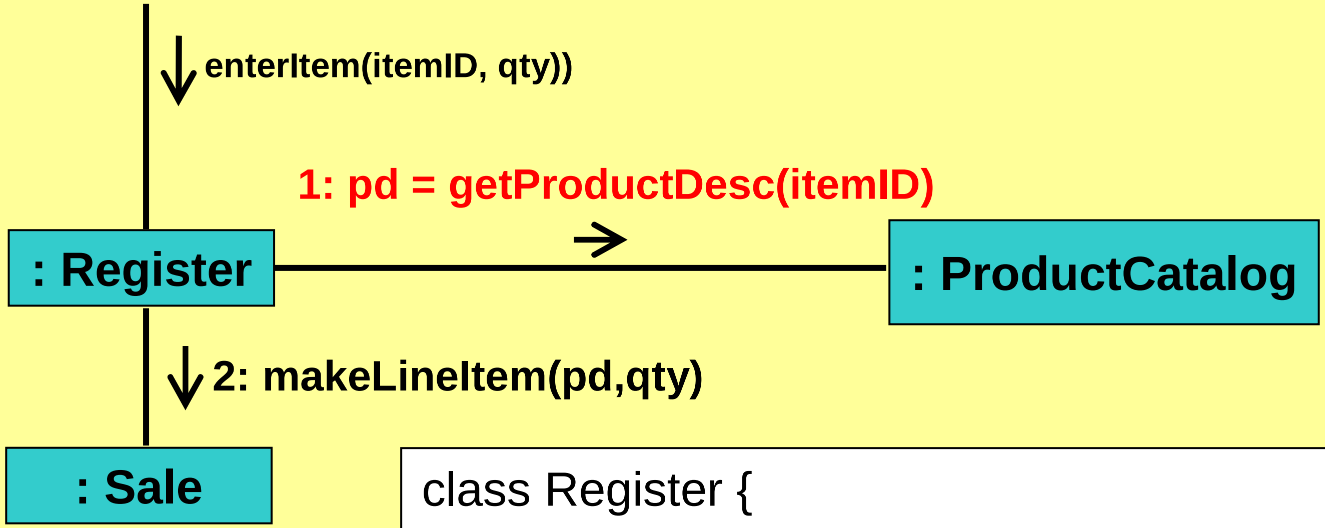
CREATING METHODS FROM INTERACTION DIAGRAMS

- The sequence of messages in an interaction diagram translates to a series of statements in the method definitions.

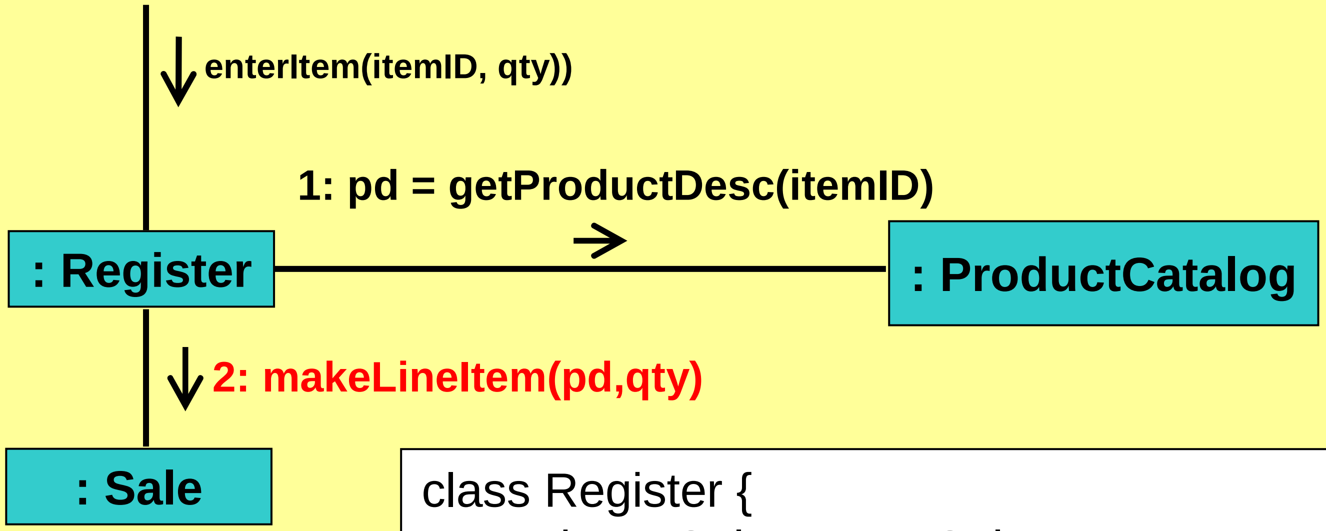
- For example, consider writing the method definition for **`enterItem`**().



```
class Register {
        private Sale currentSale;
        private ProductCatalog catalog;
        public void enterItem(ItemID itemID, int qty) {



        }
}
```



```
class Register {
        private Sale currentSale;
        private ProductCatalog catalog;
        public void enterItem(ItemID itemID, int qty) {
                ProductDescription pd;
                pd = catalog.getProductDesc(itemID);

        }
}
```



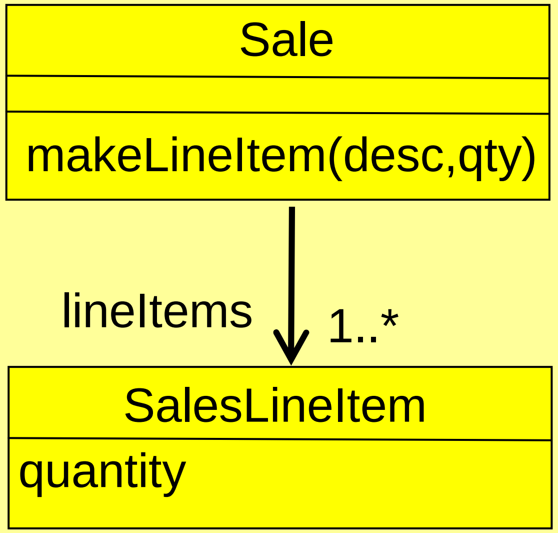
```
class Register {
        private Sale currentSale;
        private ProductCatalog catalog;
        public void enterItem(ItemID itemID, int qty) {
                ProductDescription pd;
                pd = catalog.getProductDesc(itemID);
                currentSale.makeLineItem(pd, qty);
        }
}
```

COLLECTION CLASSES

- One-to-many relationships are common. For example, a Sale is associated with a group of SalesLineItem objects.
- In OO programming languages, these relationships are usually implemented using collection objects such as Vectors, Lists, Maps, arrays and so on.

- For example, consider

```
class Sale {
        private SalesLineItem[ ] lineItems;
        public void makeLineItem(…) { }
}
```



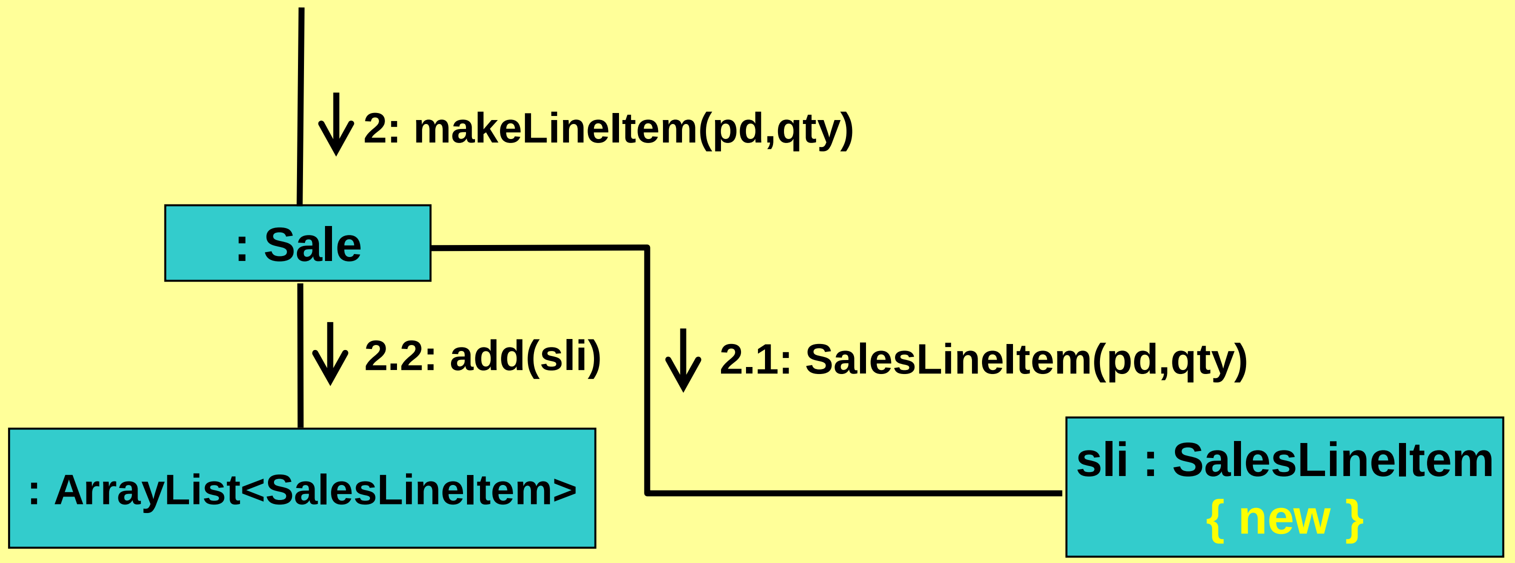
using an array object
to implement
group of **`SalesLineItem`**s

```
class Sale {
        private Vector lineItems;
        public void makeLineItem(…) { }
}
```

using a Vector object
to implement
group of **`SalesLineItem`**s

MESSAGES TO COLLECTIONS

- For example, consider the following:



- If the group of **`SalesLineItem`** objects is implemented using an array,

```
class Sale {
        private SalesLineItem[ ] lineItems;
        private int numLineItems;
        public void makeLineItem(ProductDescription desc, int qty) {
                lineItems[numLineItems] = new SalesLineItem(desc, qty);
                numLineItems++;
        }
}
```

- If the group of **`SalesLineItem`** objects is implemented using a Vector object,

```
class Sale {
        private Vector lineItems;
        public void makeLineItem(ProductDescription desc, int qty) {
                lineItems.addElement(new SalesLineItem(desc, qty));
        }
}
```

CHANGES DURING IMPLEMENTATION

- Remember! Expect and plan for changes and deviations from the design during programming.
- Realistically, the results generated during design modelling are an incomplete first step; during programming and testing, lots of changes will be made and detailed problems will be uncovered and resolved.