Detailed Class Diagram

Overview

- Design Class Diagrams (DCDs)
  - Type information
  - Accessibility
  - Visibility
  - Attributes
  - Methods
- Mapping design to code

Design Class Diagrams

- Differences from Conceptual Class Diagrams in Domain model
  - Contain types, directed associations with multiplicities, numbered actions
  - Provide visibility between objects

Type Information

- Types of attributes
- Types of method parameters/returns (can be omitted)

Accessibility of Methods and Fields

- public: can be accessed by any code
  - UML notation: +foo
- private: can be accessed only by code inside the class
  - UML notation: -foo
- protected: can be accessed only by code in the class and in its subclasses
  - UML notation: #foo
- Fields usually are not public, but have getters and setters instead

Visibility between Objects

- If object *A* sends a message to object B, then B must be visible to *A*
  - i.e., *A* should have access to a reference (pointer) to B

Attribute & Parameter Visibility

- Reference to B is an attribute of A
  - Relatively permanent: often exists for the lifetime of the objects (common)
  - E.g., Register needs to send getSpec(id) to ProductCatalog

```
class Register {
  private ProductCatalog catalog; ... }
```

- Reference to B is a parameter to a method of A
  - Relatively temporary: exists only for the scope of the method

Local Visibility

- B is a local object within a method of A
  - The object is created inside the method
  - Relatively temporary: only exists within the scope of the method
  - E.g., the subsum(subsum = s.getSubTotal();) inside getTotal() method

Global Visibility

- B is defined in a scope that encloses A's scope
  - E.g., a static field is "global" for all methods inside its declaring class
  - Relatively permanent: typically persists as long as A and B exist
  - Should be used cautiously: may violate the principles of object orientation
  - Should use Singleton pattern instead

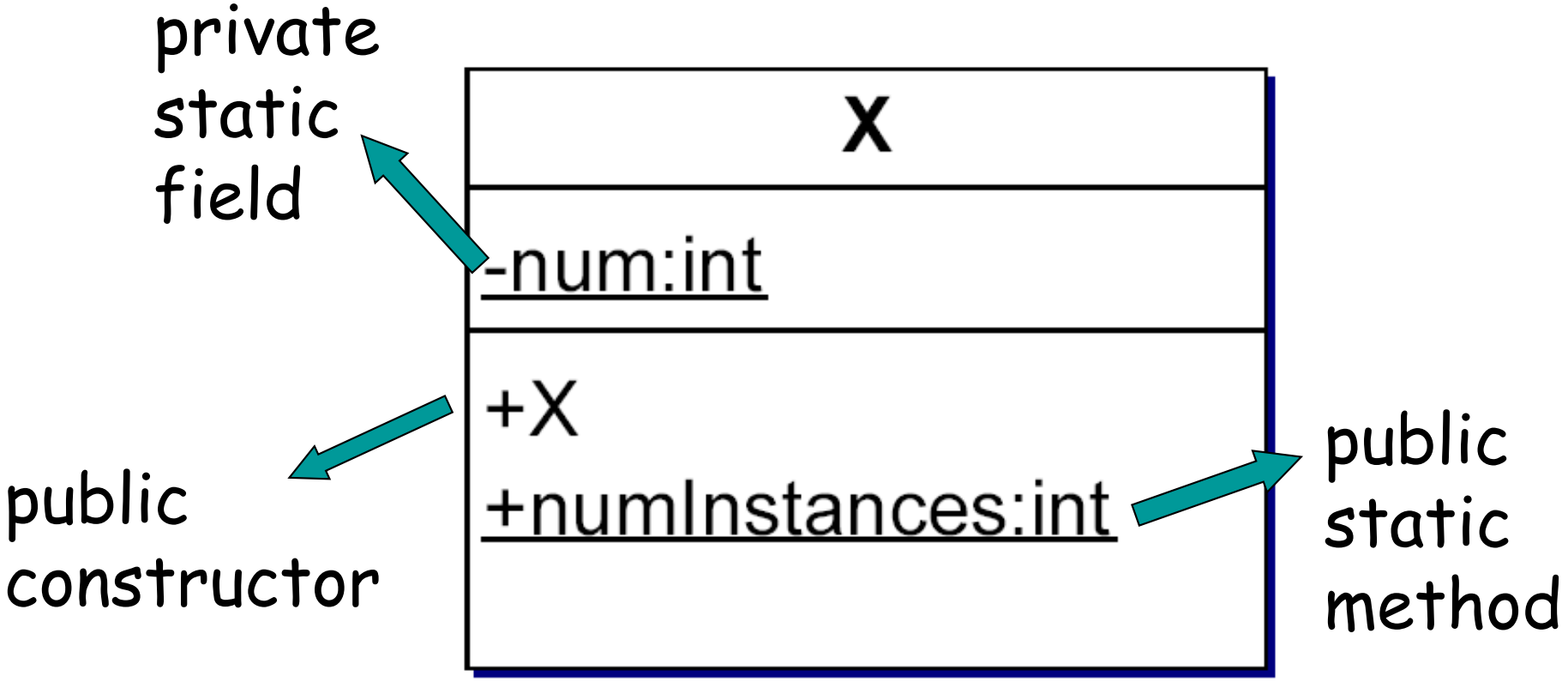
"create" messages

- create messages:
  - Language-independent
  - No create methods in the design classes
- For many languages: constructor(s)
  - Sometimes people do not show constructors in the DCD: to reduce the clutter

getters and setters for attributes

- For non-public fields
  - E.g., for price attribute of type Money
    - getPrice(): Money
    - setPrice(amt: Money)
- Methods are typically not shown in DCD

UML Class Diagram

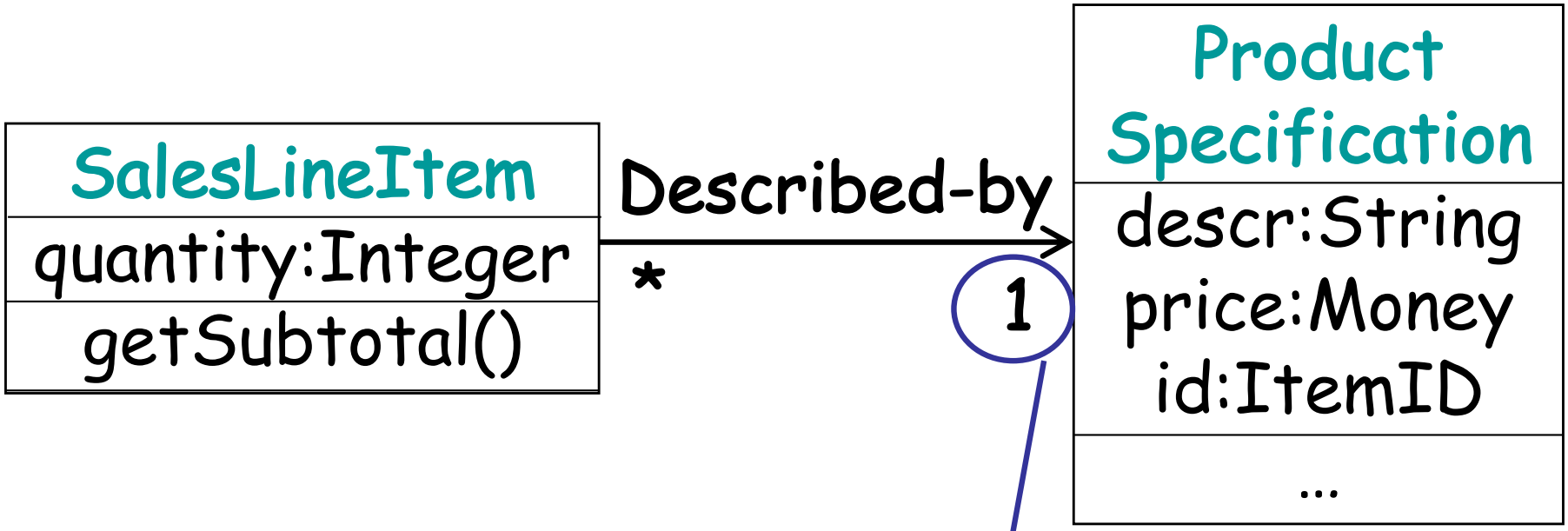


note: "static constructor"
is meaningless: by definition,
a constructor is invoked on an object

Mapping Design to Code

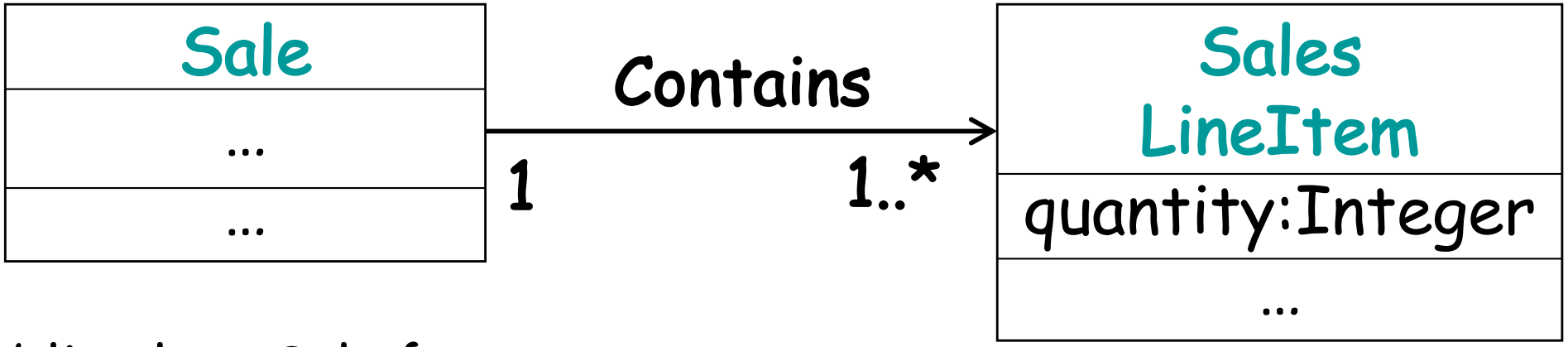
- DCDs -> classes in code
  - DCD: class names, methods, attributes, superclasses, associations, etc.
  - Tools can do this automatically
- Interaction diagrams -> method bodies
  - Interactions in the design model imply that certain method calls should be included in a method's body

Mapping Associations (* : 1, 1 : 1)



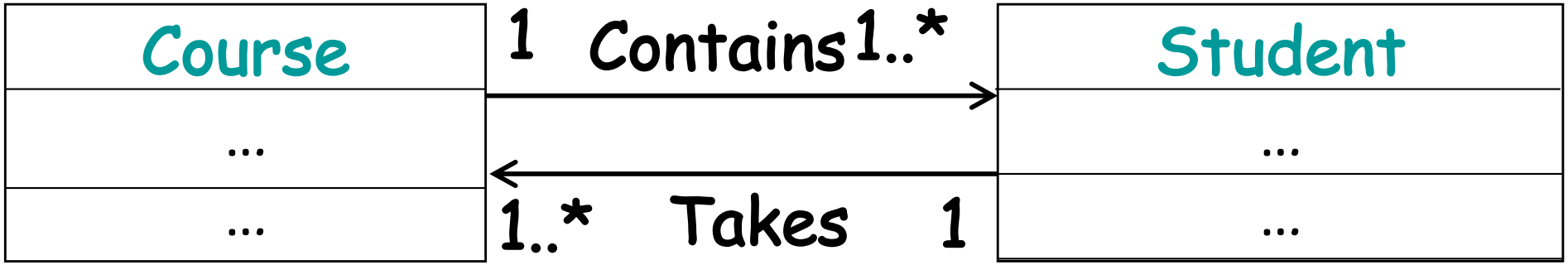

```
public class SalesLineItem {
    private int quantity;
    private ProductSpecification productSpec;
    public SalesLineItem(ProductSpecification s, int q) {…}
}
```

Mapping Associations (1 : *)




```
public class Sale {
    private List<SalesLineItem> lineItems = new
        ArrayList<SalesLineItem>();
    private Date date = new Date();
    public void makeLineItem(ProductDescription desc, int qnty)
    {
        lineItems.add(new SalesLineItem(desc, qnty));
    }
    ...
}
```

Mapping Associations (* : *)




```
public class Course {
    private List<Student> students = new ArrayList<Student>();
    public addStudent(int sid) {…}
}

public class Student {
    private List<Course> courses = new ArrayList<Course>();
    public addCourse(int cid) {…}
}
```