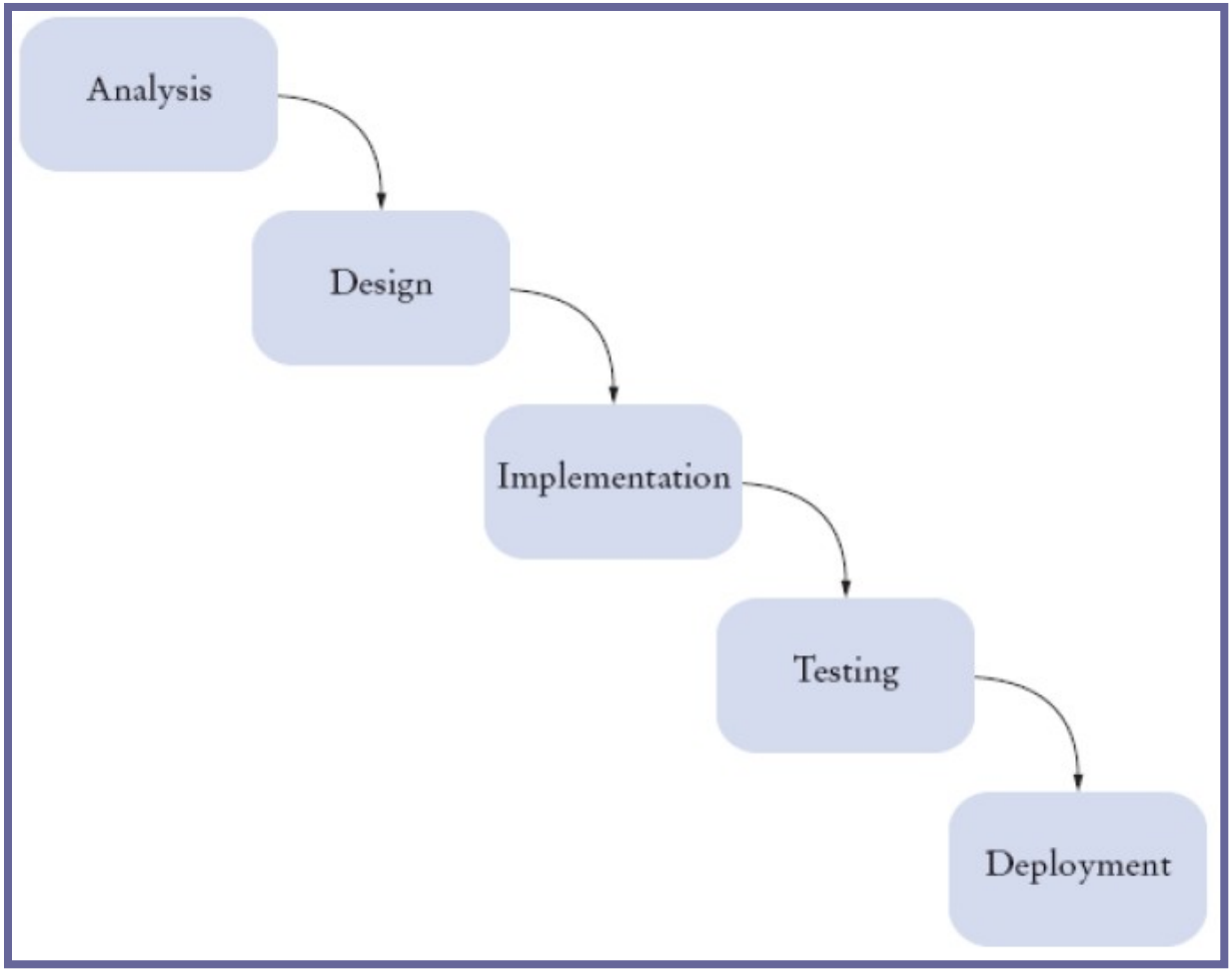
Objective

Object-Oriented Design

- To learn how to discover new classes and methods
- To understand the use of CRC cards for class discovery
- To be able to identify inheritance, aggregation, and dependency relationships between classes
- To learn how to use object-oriented design to build complex programs

Study Chapter 12 of BiG Java: Early Object 6th edition.

The Waterfall Model



Programmer Productivity

What does programmer productivity means?

Good judgment, experience, broad knowledge, attention to details, and superior planning are as important mental brilliant.

If a program takes 100-man-month, can 100 programmers finish the task in one month?

What about 10 programmers in 10 month?

There is no alternative to team work.

Object-Oriented Design

You should carry out the following tasks when you use object-oriented design.

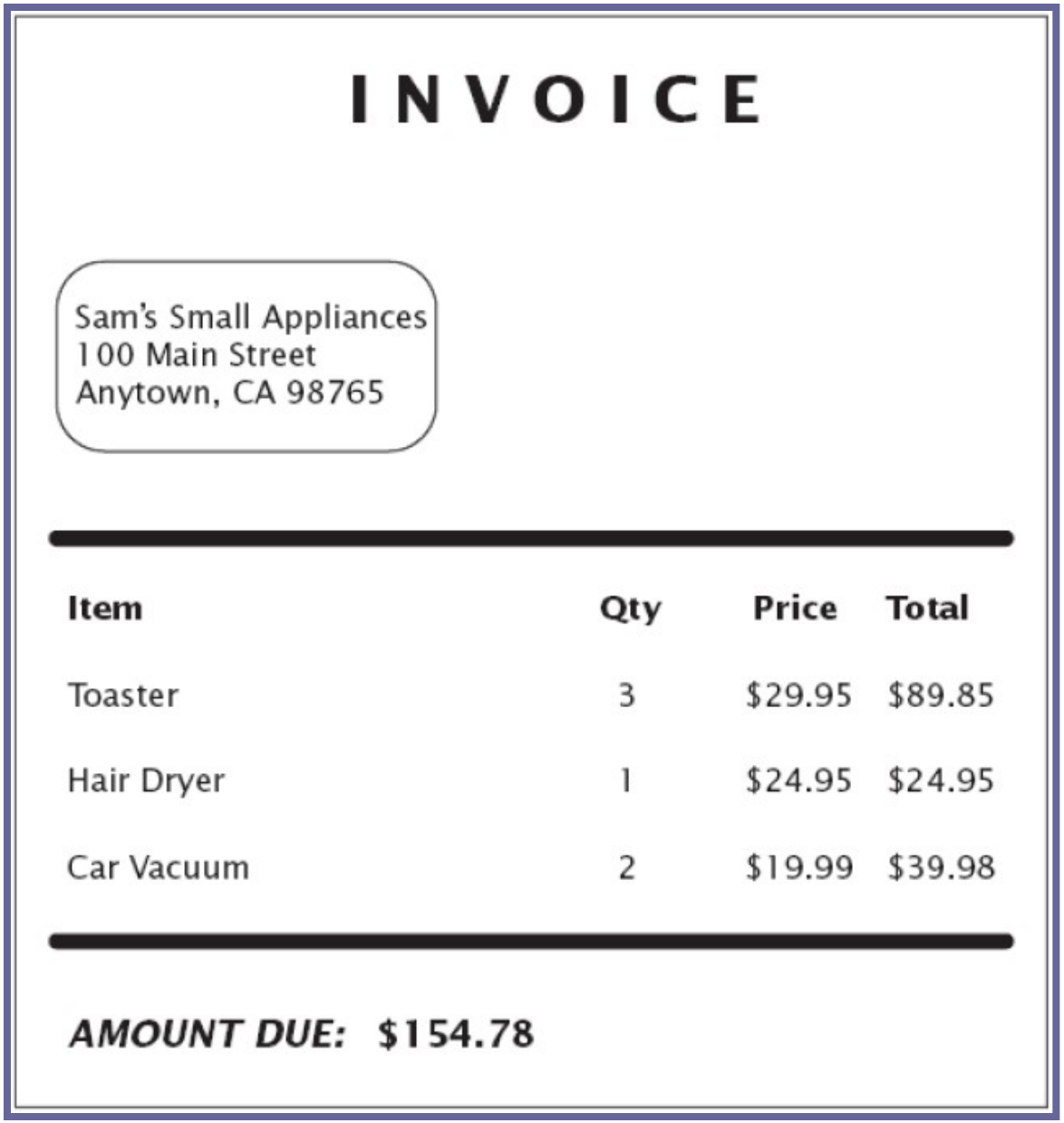
- Discover classes
- Determine responsibilities of each class
- Describe relationships between the classes

Discovering Classes

- A class represents some useful concept
- Concrete entities: bank accounts, ellipses, and products
- Abstract concepts: streams and windows
- Find classes by looking for nouns in the task description
- Define the behavior for each class
- Find methods by looking for verbs in the task description

Example: Invoice

Which classes
come to your mind?
Invoice
Customer
item
line item



INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: **$154.78**

Example: Invoice

- Good idea to keep a list of candidate classes
- Brainstorm, simply put all ideas for classes onto the list
- You can cross not useful ones later

Printing an Invoice – CRC Cards

- Discover classes
- Nouns are possible classes

```
Invoice
Address
LineItem
Product
Description
Price
Quantity
Total
Amount Due
```

Printing an Invoice – CRC Cards

- Analyze classes

```
Invoice
Address
LineItem    // Records the product and the quantity
Product
Description // Field of the Product class
Price       // Field of the Product class
Quantity    // Not an attribute of a Product
Total       // Computed–not stored anywhere
Amount Due  // Computed–not stored anywhere
```

Continued…

Printing an Invoice – CRC Cards

- Classes after a process of elimination

```
Invoice
Address
LineItem
Product
```

Finding Classes

Keep the following points in mind:

- Class represents set of objects with the same behavior
  - Entities with multiple occurrences in problem description are good candidates for objects
  - Find out what they have in common
  - Design classes to capture commonalities
- Represent some entities as objects, others as primitive types
  - Should we make a class Address or use a String?

Finding Classes

- Not all classes can be discovered in analysis phase. Like what?
- Some classes may already exist

Once classes are has been identified:

- What method each class needs to carry out.
- Look at the verbs in the task

Example: Invoice

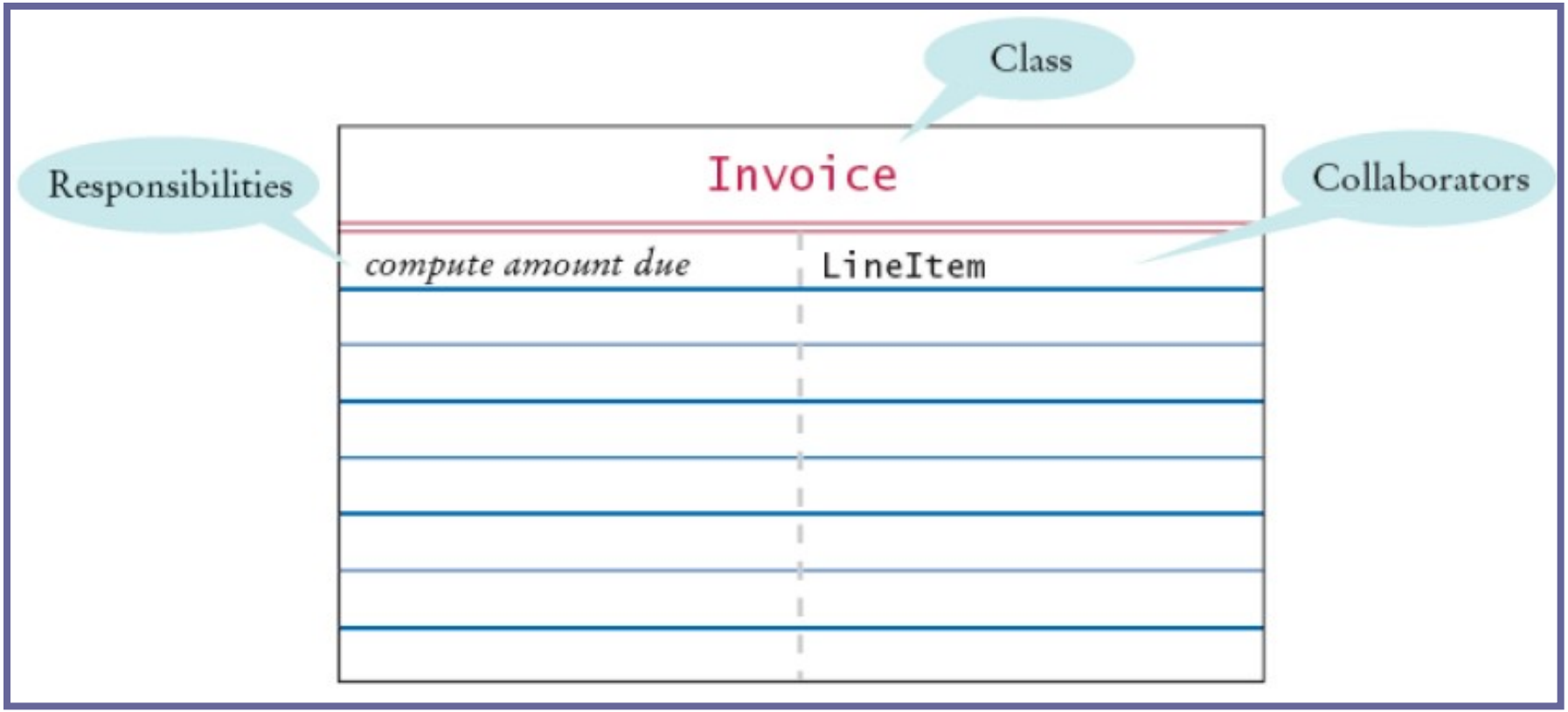
- A class needs to compute the amount due.
- Which class needs to computer amount due?
  - Do customers compute what they owe?
  - Do invoice total up the amount due?
  - Do the items total themselves up?

Best choice: Invoice Class

CRC Card

- CRC Card
- Describes a Class, its Responsibilities, and its Collaborators
- Use an index card for each class
- Pick the class that should be responsible for each method (verb)
- Write the responsibility onto the class card
- Indicate what other classes are needed to fulfill responsibility (collaborators)

CRC Card



Now look at the LineItem class. Dose it have a “get total price” method? If not add one.

CRC Cards for Printing Invoice

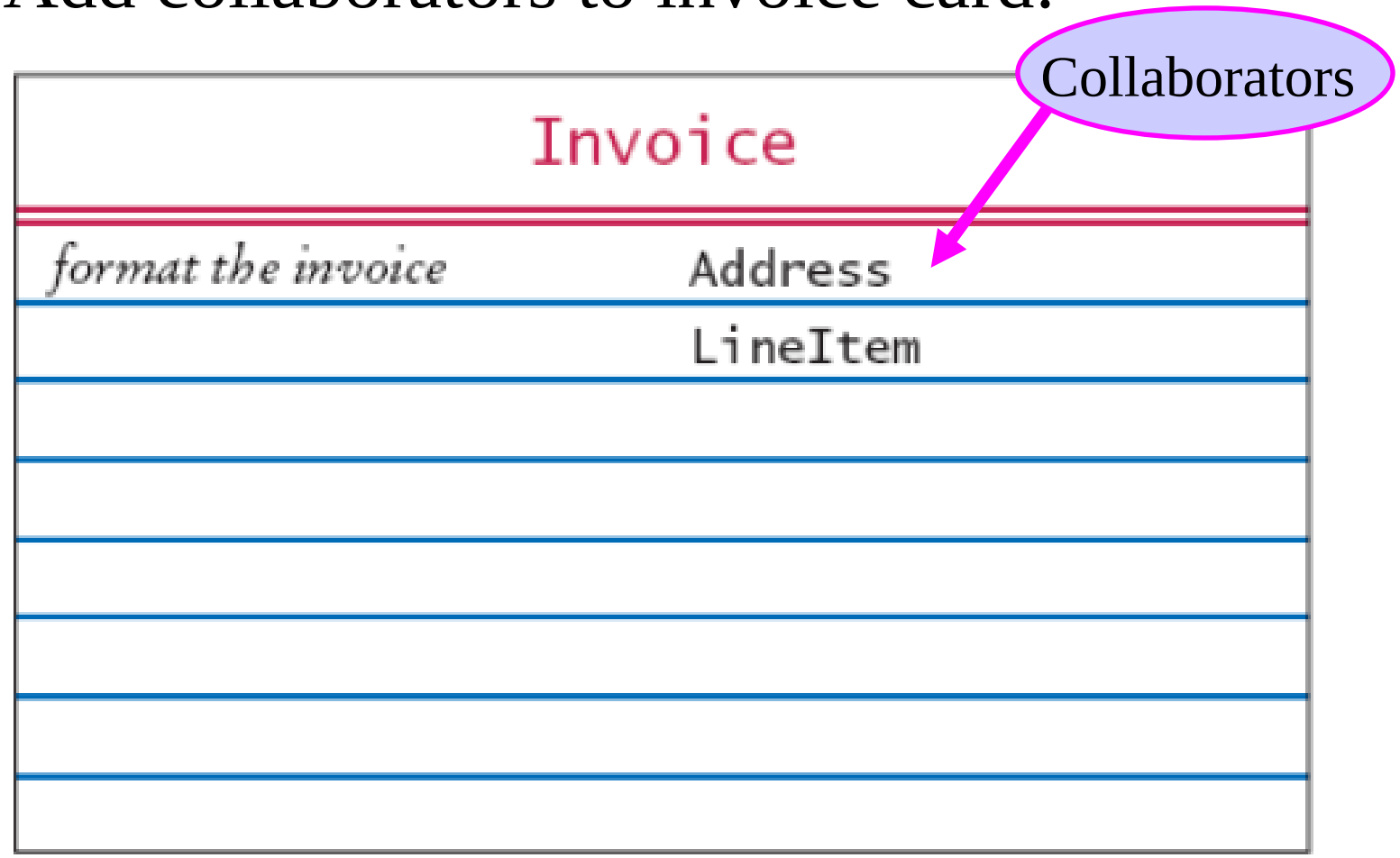
- `Invoice` and `Address` must be able to format themselves:

Invoice
format the invoice

Address
format the address

CRC Cards for Printing Invoice

- Add collaborators to invoice card:



CRC Cards for Printing Invoice

- `Product` and `LineItem` CRC cards:

Product
get description
get unit price

LineItem	
format the item	Product
get total price	

CRC Cards for Printing Invoice

- `Invoice` must be populated with products and quantities:

Invoice	
format the invoice	`Address`
add a product and quantity	`LineItem`
	`Product`

CRC Card

- How do you know you are on the right track?

Useful tip: group the cards on a table so that the collaborators are close to each other.

- Then trace the task.

Note that CRC cards are on a high level, and they are <u>informal</u>.

- Once you find that you have settled on a good set of classes
  - can you find classes with common properties? (super classes)
  - Can you organize them into clusters that are independent from each other.

Self Check

1. Suppose the invoice is to be saved to a file. Name a likely collaborator.
2. What do you do if a CRC card has ten responsibilities?

Answers

1. `FileWriter`
2. Reword the responsibilities so that they are at a higher level, or come up with more classes to handle the responsibilities.

Relationships Between Classes

If classes have common behavior arrange them in super and sub classes.

If classes are not related, you can assign them to different programmers.

Relationships Between Classes

- Inheritance (is-a relationship)
- Aggregation (has-a relationship)
- Dependency (use relationship)

Inheritance

- *Is-a* relationship
- Relationship between a more general class (superclass) and more specialized class (subclass)

Example:

- Every savings account is a bank account
- Every circle is an ellipse (with equal width and height)

Continued…

Inheritance

- Inheritance is sometimes abused
  - Should the class Tire be a subclass of a class Circle?
  - Thought it may be convenient for a programmer, but remember that tires are car parts while circles are geometric objects.
- However, there is a relationship between tire and circle.
  - A tire has a circle as its boundary.
  - – The *has-a* relationship would be more appropriate

Aggregation

- *Has-a* relationship.
- Each Tire aggregates a Circle object.
- Objects of one class contain references to objects of another class
- Use an instance variable
  - A tire has a circle as its boundary:

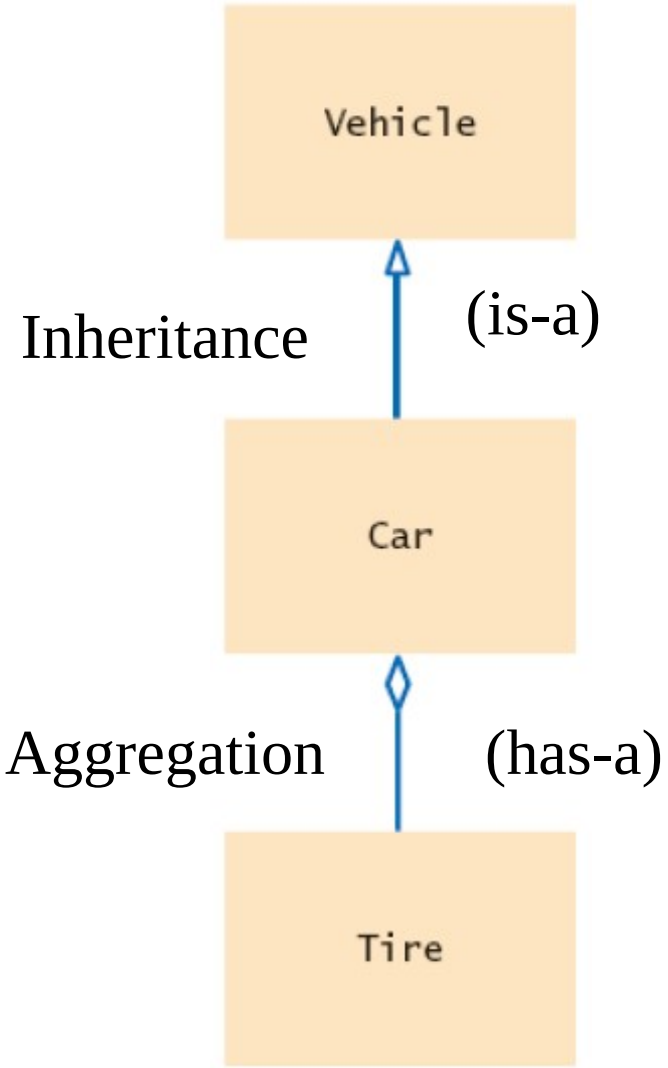
```
class Tire
{
    . . .
    private String rating;
    private Circle boundary;
}
```

- Every car has a tire (in fact, it has four)

Example

```
class Car extends Vehicle
{
   . . .
   private Tire[] tires;
}
```

```
class Tire
{
   . . .
   private String rating;
   private Circle boundary;
}
```



UML Notation for Inheritance and Aggregation

Dependency

- *Uses* relationship
- Example: many of our applications depend on the Scanner class to read input
- Aggregation is a stronger form of dependency

What is difference between aggregation and dependency?

- Use aggregation to remember another object between method calls. (An instance of object)

UML Relationship Symbols

Relationship	Symbol	Line Style	Arrow Tip
Inheritance		Solid	Triangle
Interface Implementation		Dotted	Triangle
Aggregation		Solid	Diamond
Dependency		Dotted	Open
Inner Class		Solid	Circle

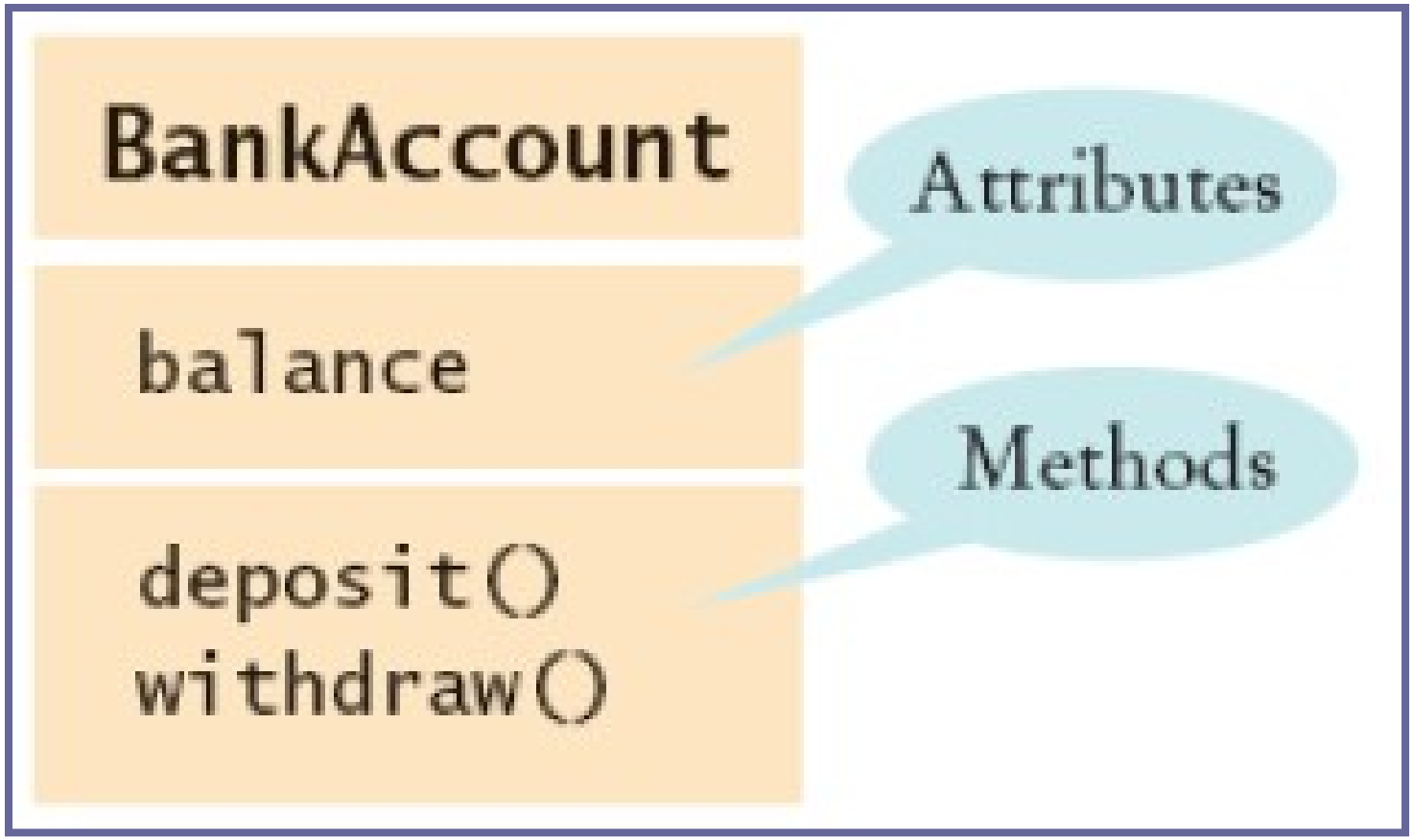
Review

Before writing code for a complex problem, you need to design a solutions.

Steps of design:

- Discover classes
- Determine responsibilities of each class
- Describe relationships between the classes

Attributes and Methods in UML Diagrams



Attributes and Methods in a Class Diagram

Visibility

+ Public
- Private
Protected
~ Package
Show static members underlined.

Abstract class in UML

Multiplicities

- any number (zero or more): *
- one or more: 1..*
- zero or one: 0..1
- exactly one: 1

An Aggregation Relationship with Multiplicities

Aggregation and Association

- Association: more general relationship between classes
- Use early in the design phase
- A class is associated with another if you can navigate from objects of one class to objects of the other
- Given a `Bank` object, you can navigate to `Customer` objects

Continued...

Aggregation and Association

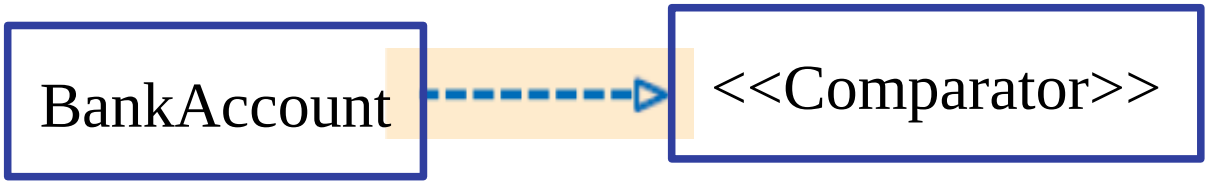
Association is useful in early design process.

An Association Relationship

Interface

```
class BankAccount implements Comparator {

}
```



Five-Steps Development Process

- Gather requirements
- Use CRC cards to find classes, responsibilities, and collaborators
- Use UML diagrams to record class relationships
- Use `javadoc` to document method behavior
- Implement your program

Suggested Exercises

R12.2, R12.5, R12.12

Design the following programs and develop their UML diagrams.

E12.5, P12.6