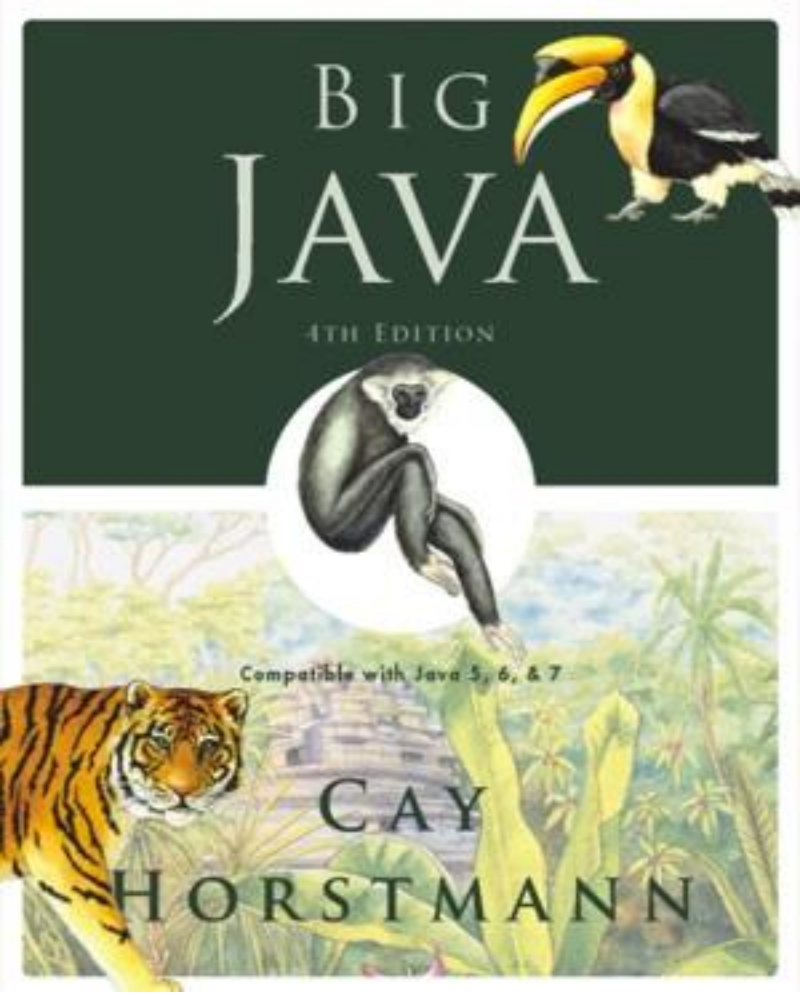


الفصل ١٢ - تصميم الكائنية الموجهة

Chapter 12 – Object-Oriented Design

Chapter Goals

- To learn about the software life cycle
- 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 master the use of UML class diagrams to describe class relationships
- To learn how to use object-oriented design to build complex programs

- لمعرفة المزيد عن دورة حياة البرمجيات
- لمعرفة كيفية اكتشاف الطبقات وأساليب جديدة
- لفهم استخدام بطاقات CRC لاكتشاف الطبقة
- لتكون قادرة على تحديد الميراث، والتجميع، وعلاقات التبعية بين الطبقات
- لإتقان استخدام الرسوم البيانية فئة UML لوصف العلاقات الدرجة
- لمعرفة كيفية استخدام تصميم وجوه المنحى لبناء البرامج المعقدة

The Software Life Cycle

- Encompasses all activities from initial analysis until obsolescence
- Formal process for software development
  - *Describes phases of the development process*
  - *Gives guidelines for how to carry out the phases*
- Development process
  - *Analysis*
  - *Design*
  - *Implementation*
  - *Testing*
  - *Deployment*

- يشمل جميع الأنشطة من التحليل الأولي حتى التقادم
- عملية رسمية لتطوير البرمجيات
- يصف مراحل عملية التنمية
- يعطي مبادئ توجيهية لكيفية تنفيذ المراحل

- عملية التنمية
- تحليل
- تصميم
- تطبيق
- تجريب
- نشر

Analysis

- Decide what the project is supposed to do
- Do not think about how the program will accomplish tasks
- Output: Requirements document
  - *Describes what program will do once completed*
  - *User manual: Tells how user will operate program*
  - *Performance criteria*

- تقرر ما يفترض المشروع أن تفعل
- لا نفكر في كيفية البرنامج سوف تنجز المهام
- الإخراج: متطلبات وثيقة
- يصف ماذا ستفعل عندما أكملت البرنامج
- دليل المستخدم: يروي كيف المستخدم سيعمل البرنامج
- معايير الأداء

Design

تصميم

- Plan how to implement the system
- Discover structures that underlie problem to be solved
- Decide what classes and methods you need
- Output:
  - *Description of classes and methods*
  - *Diagrams showing the relationships among the classes*

- التخطيط لكيفية تنفيذ النظام
- هياكل اكتشاف التي تكمن وراء المشكلة التي يتعين حلها
- تقرر ما الطبقات والأساليب التي تحتاج
- الإخراج:
- وصف الفئات وأساليب
- الرسوم البيانية التي تبين العلاقات بين الطبقات

Implementation

تطبيق

- Write and compile the code
- Code implements classes and methods discovered in the design phase
- Program Run: Completed program

- كتابة وترجمة التعليمات البرمجية
- كود تطبق الفئات وأساليب اكتشافه في مرحلة التصميم
- برنامج تشغيل: برنامج مكتمل

Testing

تجريب

- Run tests to verify the program works correctly
- Program Run: A report of the tests and their results

- اختبارات للتحقق من تشغيل يعمل البرنامج بشكل صحيح
- برنامج تشغيل: تقرير عن الاختبارات ونتائجها

Deployment

نشر

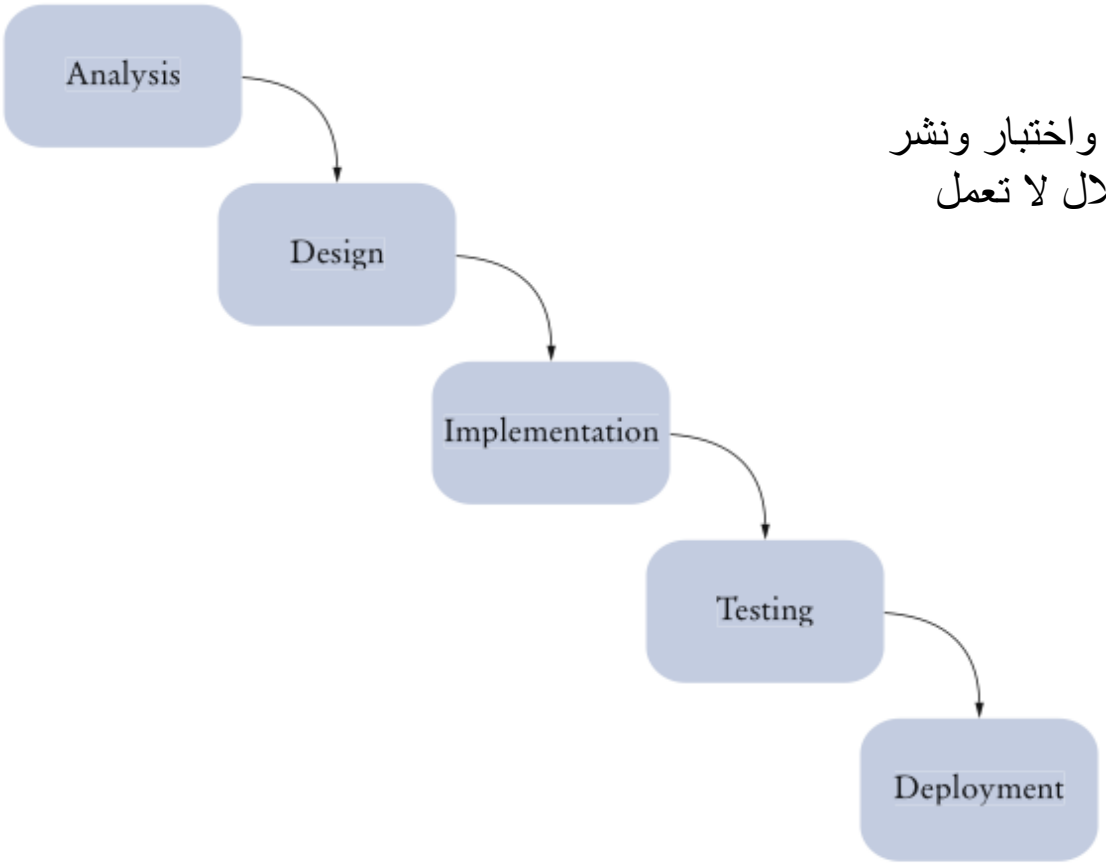
- Users install program
- Users use program for its intended purpose

للمستخدمين تثبيت برنامج
للمستخدمين استخدام البرنامج للغرض المقصود

The Waterfall Model

الشلال نموذج

- Sequential process of analysis, design, implementation, testing, and deployment
- When rigidly applied, waterfall model did not work



- عملية متسلسلة من تحليل وتصميم وتنفيذ واختبار ونشر
- عندما يطبق بشكل صارم، لم نموذج الشلال لا تعمل

Figure 1 The Waterfall Model

The Spiral Model

والنموذج اللولبي

- Breaks development process down into multiple phases
- Early phases focus on the construction of *prototypes*
- Lessons learned from development of one prototype can be applied to the next iteration

- فواصل عملية التنمية وصولا إلى مراحل متعددة
- وتركز المراحل الأولى على بناء النماذج
- الدروس المستفادة من تطوير نموذج أولي واحد يمكن تطبيقها على التكرار التالي

The Spiral Model

والنموذج اللولبي

- Problem: Can lead to many iterations, and process can take too long to complete

- المشكلة: يمكن أن يؤدي إلى الكثير من التكرار، والعملية يمكن أن تستغرق وقتا طويلا جدا لإكمال

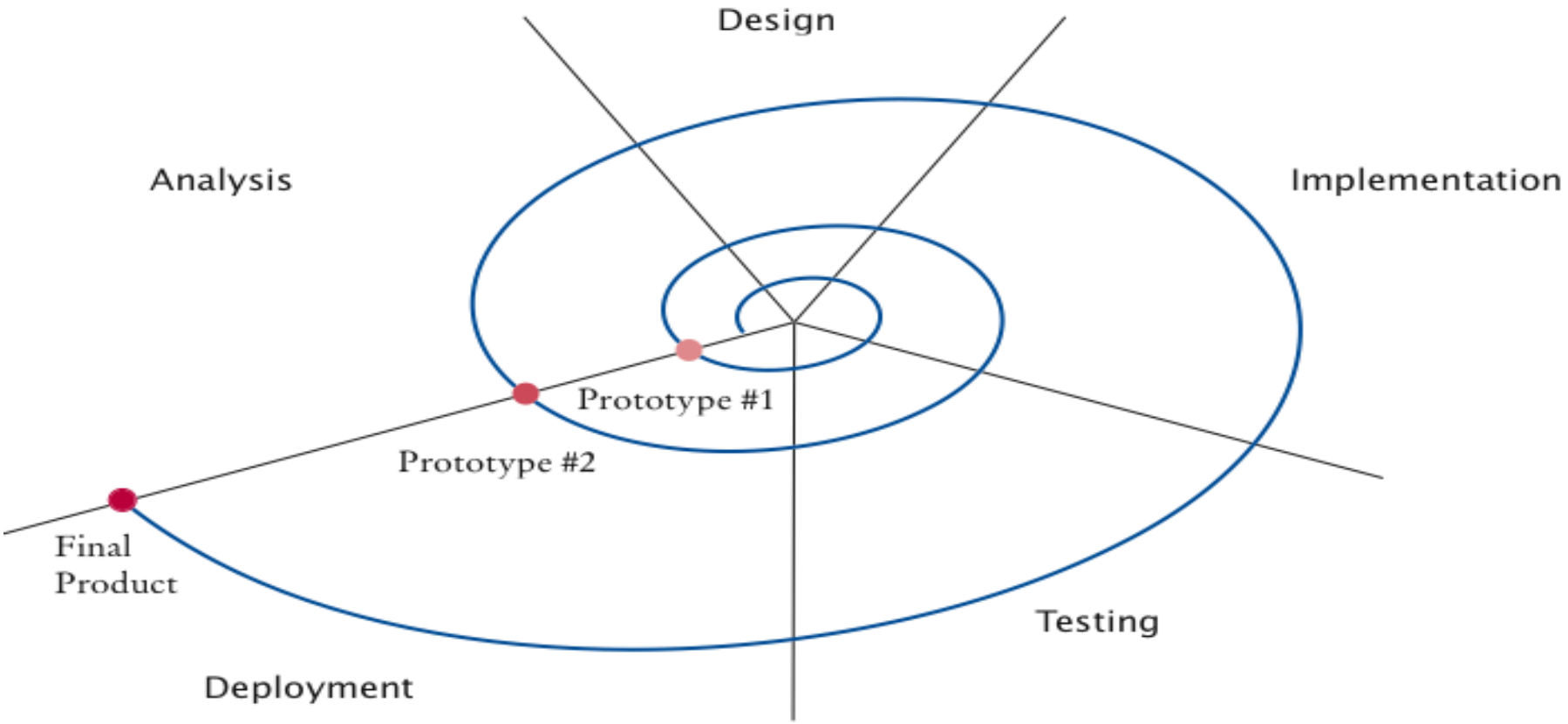


Figure 2 A Spiral Model

Activity Levels in the Rational Unified Process

مستويات النشاط في
عملية الرشيد الموحدة

Development process methodology by the inventors of UML

منهجية عملية التنمية من قبل المخترعين من UML

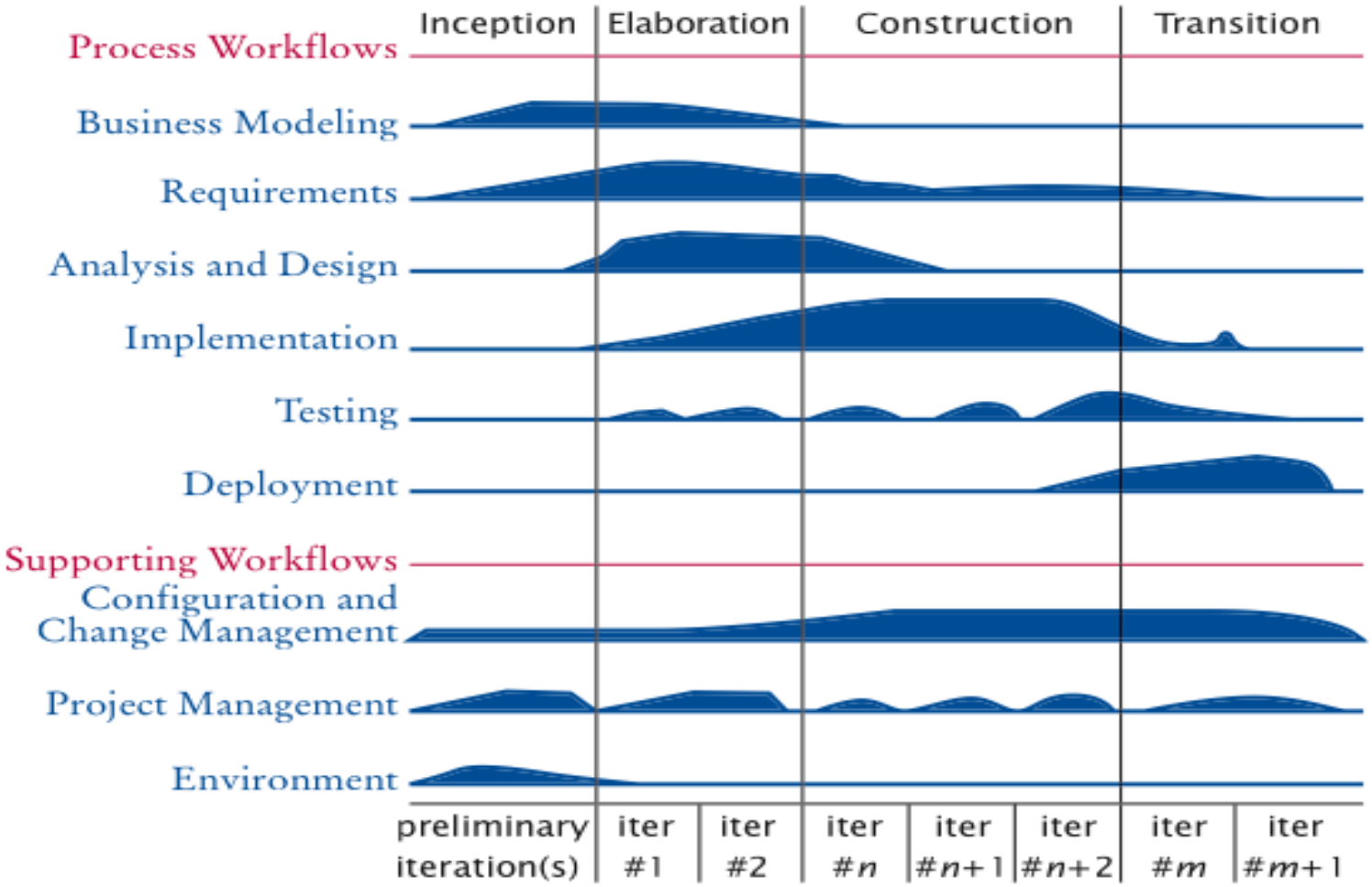


Figure 3 Activity Levels in the Rational Unified Process Methodology

Extreme Programming

- Strives for simplicity
- Removes formal structure
- Focuses on best practices

- تسعى للبساطة
- يزيل هيكل رسمي
- يركز على أفضل الممارسات

Extreme Programming

البرمجة المتطرفة

- Realistic planning
  - *Customers make business decisions*
  - *Programmers make technical decisions*
  - *Update plan when it conflicts with reality*
- Small releases
  - *Release a useful system quickly*
  - *Release updates on a very short cycle*
- Metaphor
  - *Programmers have a simple shared story that explains the system*

- التخطيط الواقعي
- العملاء على اتخاذ القرارات التجارية
- المبرمجين جعل القرارات الفنية
- تحديث الخطة عندما يتعارض مع الواقع
- النشرات الصغيرة
- الافراج عن نظام مفيد بسرعة
- الافراج عن التحديثات على دورة قصيرة جدا
- استعارة
- المبرمجين لديهم قصة مشتركة البسيطة التي يفسر النظام

Extreme Programming

- Simplicity
  - *Design as simply as possible instead of preparing for future complexities*
- Testing
  - *Programmers and customers write test cases*
  - *Test continuously*
- Refactoring
  - *Restructure the system continuously to improve code and eliminate duplication*

- البساطة
- تصميم ببساطة ممكن بدلا من التحضير لالتعقيدات المستقبلية
- تجريب
- المبرمجين والعملاء إرسال حالات الاختبار
- اختبار مستمر
- إعادة بيع ديون
- إعادة هيكلة النظام باستمرار لتحسين رمز والتخلص من الازدواجية

Extreme Programming

- Pair programming
  - *Two programmers write code on the same computer*
- Collective ownership
  - *All programmers can change all code as needed*
- Continuous integration
  - *Build the entire system and test it whenever a task is complete*

- برمجة الزوج
- اثنين من المبرمجين كتابة التعليمات البرمجية على نفس الكمبيوتر
- الملكية الجماعية
- ويمكن لجميع المبرمجين تغيير كل رمز حسب الحاجة
- التكامل المستمر
- بناء نظام كامل واختباره كلما اكتمال المهمة

Extreme Programming

- 40-hour week
  - *Don’t cover up unrealistic schedules with heroic effort*
- On-site customer
  - *A customer is accessible to the programming team at all times*
- Coding standards
  - *Follow standards that emphasize self-documenting code*

- الأسبوع ٤٠ ساعة
- لا تستر جداول غير واقعية مع جهدا بطوليا
- على موقع العميل
- عميل يمكن الوصول إلى فريق البرمجة في جميع الأوقات
- معايير التشفير
- اتبع المعايير التي تؤكد على رمز التوثيق الذاتي

Self Check 12.1

Suppose you sign a contract, promising that you will, for an agreed-upon price, design, implement, and test a software package exactly as it has been specified in a requirements document. What is the primary risk you and your customer are facing with this business arrangement?

Answer: It is unlikely that the customer did a perfect job with the requirements document. If you don't accommodate changes, your customer may not like the outcome. If you charge for the changes, your customer may not like the cost.

افترض أنك توقيع العقد، واعدا أن صح التعبير، لمتفق عليها سعر، تصميم وتنفيذ واختبار مجموعة من البرامج تماما كما تم تحديد ذلك في وثيقة المتطلبات. ما هي مخاطر الأساسي الذي وعميلك تواجه مع هذا الترتيب العمل؟

الجواب: إنه من غير المرجح أن العميل قام بعمل مثالية مع وثيقة المتطلبات. إذا كنت لا تستوعب التغييرات، العميل الخاص بك قد لا يعجب النتيجة. إذا كنت مسؤولا عن التغييرات، العميل الخاص بك قد لا يعجب التكلفة.

Self Check 12.2

Does Extreme Programming follow a waterfall or a spiral model?

Answer: An “extreme” spiral model, with lots of iterations.

- لا تتبع البرمجة المتطرفة شلال أو نموذج دوامة؟
- الجواب: "المتطرفة" نموذج دوامة، مع الكثير من التكرار.

Self Check 12.3

What is the purpose of the “on-site customer” in Extreme Programming?

Answer: To give frequent feedback as to whether the current iteration of the product fits customer needs.

- ما هو الغرض من "العميل في الموقع" المتطرفة في البرمجة؟
- الإجابة: لإعطاء ملاحظات متكررة حول ما إذا كان التكرار الحالي للمنتج يناسب احتياجات العملاء.

Object-Oriented Design

تصميم الشيئية

1. Discover classes
2. Determine responsibilities of each class
3. Describe relationships between the classes

- اكتشاف الطبقات
- تحديد مسؤوليات كل فئة
- وصف العلاقات بين الطبقات

Discovering Classes

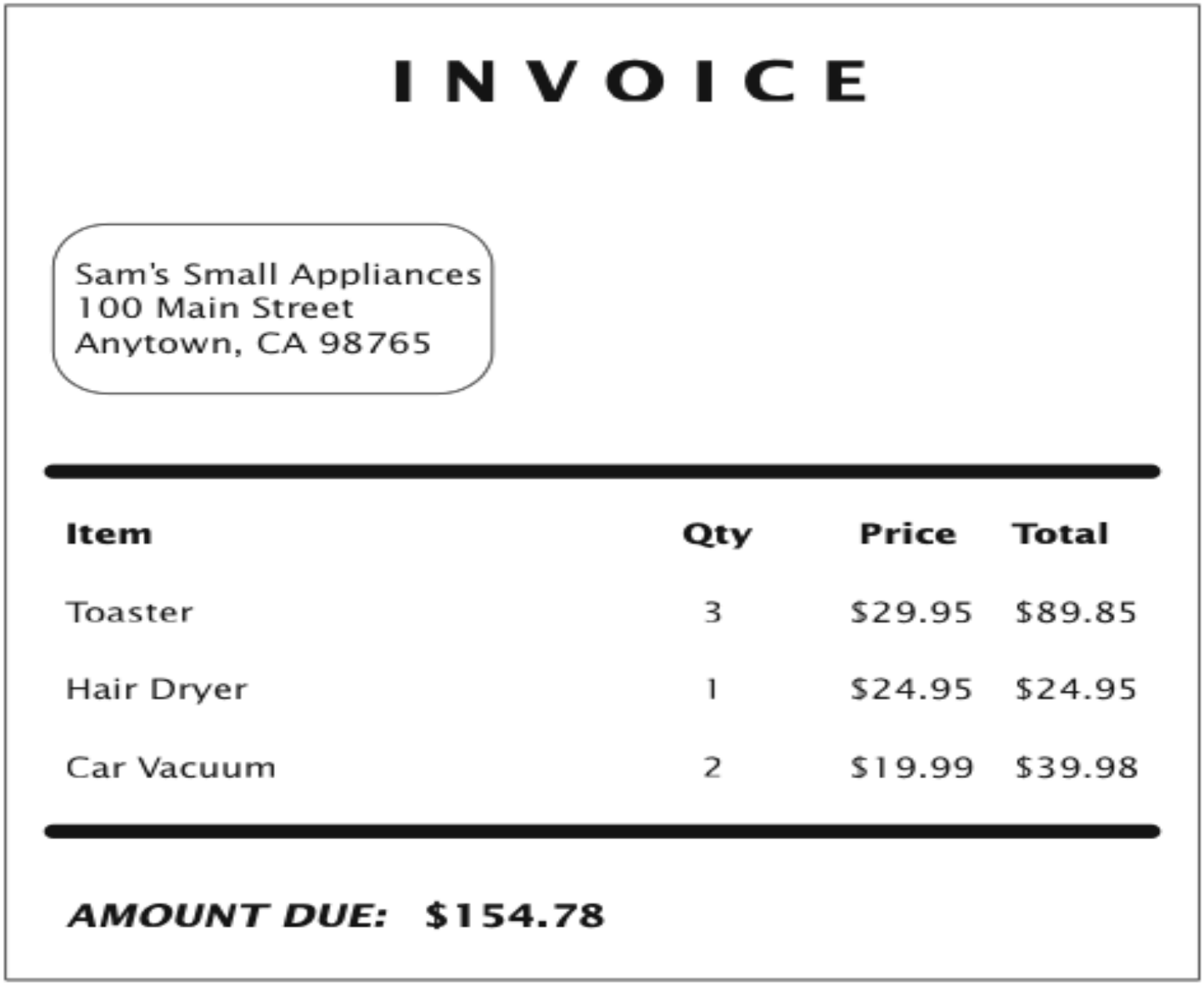
اكتشاف فئات

- 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

- وتمثل فئة بعض مفهوم مفيد
- كيانات محددة: الحسابات المصرفية، والحذف، والمنتجات
- المفاهيم المجردة: تيارات والنوافذ
- البحث الطبقات من خلال البحث عن الأسماء في وصف مهمة
- تعريف السلوك لكل فئة
- البحث عن الأساليب التي تبحث عن الأفعال في وصف المهمة

Example: Invoice

مثال: فاتورة



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**

Figure 4
An Invoice

Example: Invoice

- Classes that come to mind: `Invoice`, `LineItem`, and `Customer`
- 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

- الفئات التي تتبادر إلى الذهن: فاتورة، LineItem، والعميل
- فكرة جيدة للحفاظ على قائمة الطبقات مرشح
- العصف الذهني، ببساطة كل الأفكار لفئات على قائمة
- يمكنك عبور تلك ليست مفيدة في وقت لاحق

Finding Classes

العثور على فصول

- 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`*?*
  - *Not all classes can be discovered in analysis phase*
  - *Some classes may already exist*

- الحفاظ على النقاط التالية في الاعتبار:
- الفئة تمثل مجموعة من الكائنات مع نفس السلوك
- الكيانات مع حوادث متعددة في وصف المشكلة هي مرشحة جيدة لكائنات
- معرفة ما لديهم من القواسم المشتركة
- دروس تصميم للقبض على القواسم المشتركة
- تمثل بعض الكيانات ككائنات، والبعض الآخر عن أنواع بدائية
- وينبغي أن نجعل عنوان فئة أو استخدام سلسلة؟
- لا يمكن اكتشافها جميع الطبقات في مرحلة التحليل
- قد توجد بعض الفئات بالفعل

CRC Card

بطاقةCRC

- Describes a **C** lass, its **R**esponsibilities, and its **C**ollaborators
- 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

- يصف Cفئة، Rمسؤولياتها، وCالمتعاونين معها
- استخدام بطاقة الرقم القياسي لكل فئة
- اختيار الفئة التي يجب أن تكون مسؤولة عن كل طريقة (فعل)
- إرسال المسؤولية على بطاقة فئة

Continued

CRC Card

- Indicate what other classes are needed to fulfill responsibility (collaborators)
- تشير ما الطبقات الأخرى اللازمة لتحقيق المسؤولية (المتعاونين)

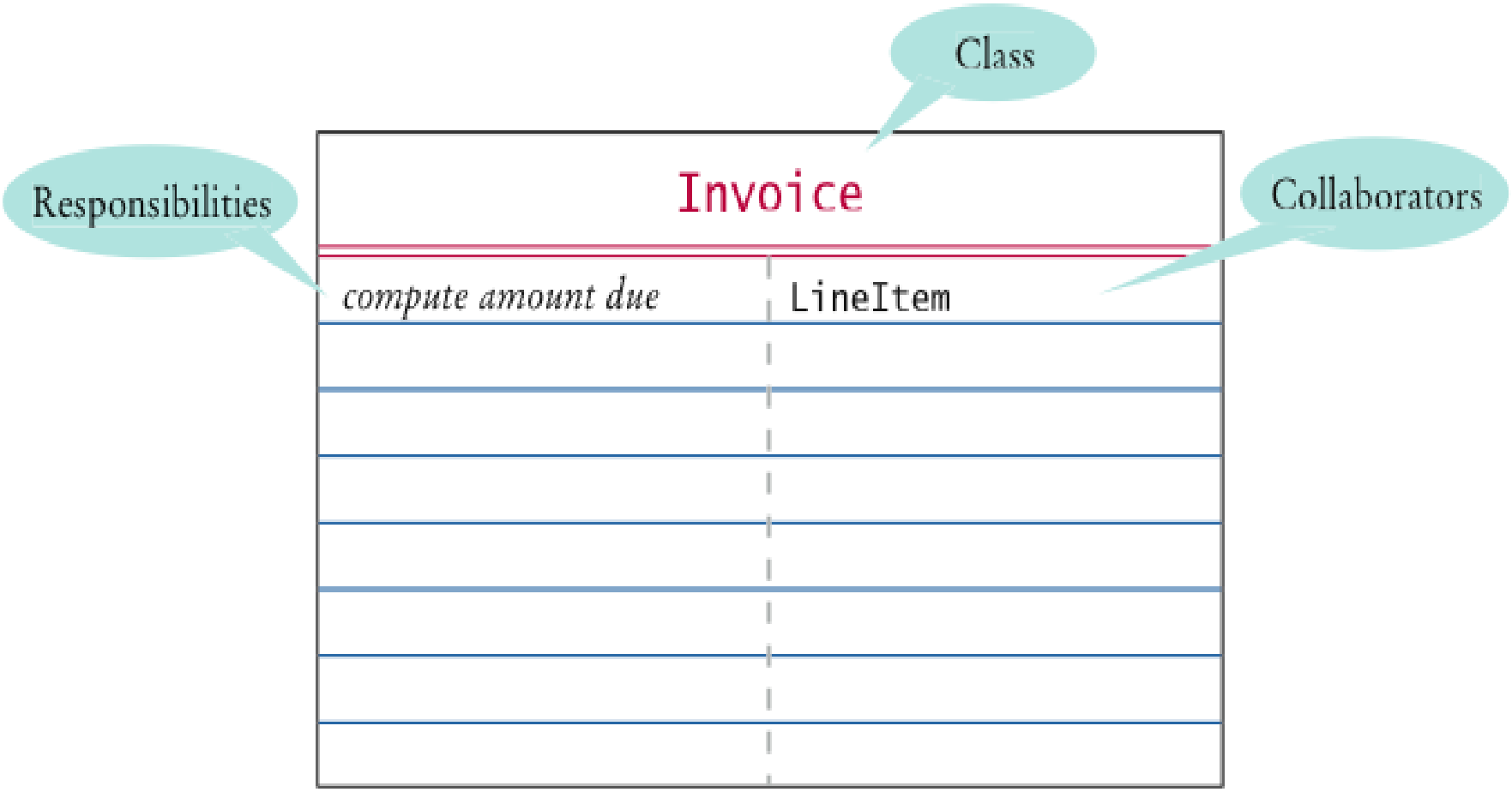


Figure 5 A CRC Card

Self Check 12.4

Suppose the invoice is to be saved to a file. Name a likely collaborator.

Answer: `PrintStream`

لنفترض أن الفاتورة ليتم حفظها إلى ملف. اسم متعاون المرجح.
الجواب: PrintStream

Self Check 12.5

Looking at the invoice in Figure 4, what is a likely responsibility of the `Customer` class?

Answer: To produce the shipping address of the customer.

- أبحث في الفاتورة في الشكل (٤)، ما هو المسؤولية المحتملة للفئة العملاء `Customer`؟
- الإجابة: لإنتاج عنوان الشحن للعميل.

Self Check 12.6

What do you do if a CRC card has ten responsibilities?

Answer: Reword the responsibilities so that they are at a higher level, or come up with more classes to handle the responsibilities.

- ماذا تفعل إذا بطاقة CRC ديها عشرة المسؤوليات؟
- الجواب: إعادة صياغة المسؤوليات بحيث تكون على مستوى أعلى، أو الخروج مع المزيد من الطبقات للتعامل مع المسؤوليات.

Relationships Between Classes

العلاقات بين الطبقات

- Inheritance
- Aggregation
- Dependency

- وراثة
- تجميع
- التبعية

Inheritance

وراثة

- *Is-a* relationship
- Relationship between a more general class (superclass) and a more specialized class (subclass)
- Every savings account is a bank account
- Every circle is an ellipse (with equal width and height)
- It is sometimes abused
  - *Should the class `Tire` be a subclass of a class `Circle`?*
    - *The has-a relationship would be more appropriate*

- هو-علاقة
- العلاقة بين فئة أعم (المتفوقة) وطبقة أكثر تخصصا (فرعية)
- كل حساب التوفير هو حساب مصرفي
- كل دائرة هو القطع الناقص (مع متساوية العرض والارتفاع)
- يساء أحيانا يكون
- يجب أن تكون الطبقة الإطارات فئة فرعية من دائرة الصف؟
- ان لديها، علاقة يكون من الأنسب

Aggregation

تجميع

- *Has-a* relationship
- 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)

- ويملك علاقة
- كائنات من فئة واحدة تحتوي على مراجع الكائنات من فئة أخرى
- استخدام متغير المثال

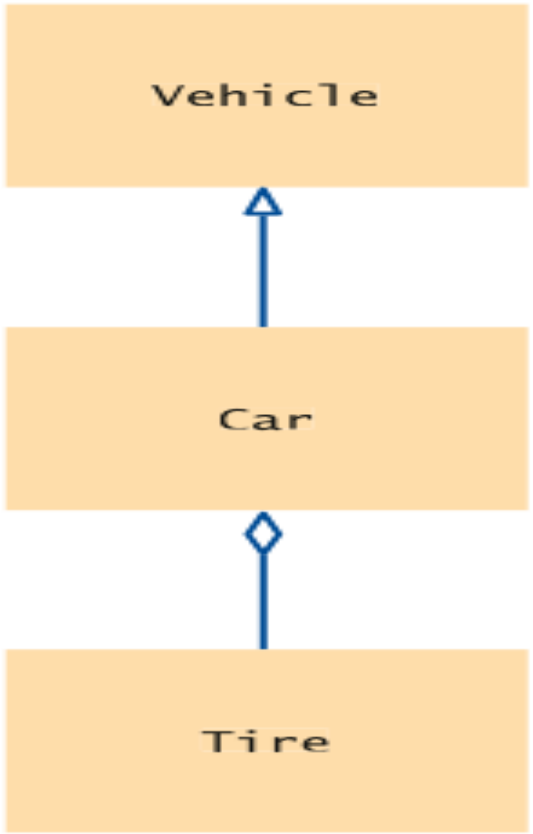
- Aالإطارات لديه دائرة كما حدودها:

- كل سيارة لديها الاطارات (في الواقع، كان لديه أربعة)

Example

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

Figure 6
UML Notation for
Inheritance and Aggregation



Dependency

التبعية

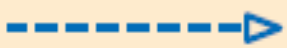
- *Uses* relationship
- Example: Many of our applications depend on the `Scanner` class to read input
- Aggregation is a stronger form of dependency
- Use aggregation to remember another object between method calls

- يستخدم العلاقة
- مثال: تعتمد كثير من تطبيقاتنا على الطبقة الماسح الضوئي لقراءة المدخلات
- التجميع هو شكل أقوى من التبعية
- استخدام التجميع أن نتذكر كائن آخر بين استدعاءات الأسلوب

UML Relationship Symbols

العلاقة رموز UML

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

علاقة	الرمز	نمط الخط	طرف المثلث
وراثة		متماسك	مثلث
تنفيذ واجهة		المنقطة	مثلث
تجميع		متماسك	ماسة
التبعية		المنقطة	مفتوح

Self Check 12.7

Consider the `Bank` and `BankAccount` classes of Chapter 7. How are they related?

Answer: Through aggregation. The bank manages bank account objects.

- النظر في فصول البنك و BankAccountالفصل ٧. كيف هم ذات الصلة؟
- الإجابة: من خلال التجميع. البنك يدير كائنات حساب البنك.

Self Check 12.8

Consider the `BankAccount` and `SavingsAccount` objects of Chapter 10. How are they related?

Answer: Through inheritance.

- النظر في BankAccountو SavingsAccountالأشياء من الفصل ١٠. كيف يتم ذات الصلة؟
- الإجابة: من خلال الميراث.

Self Check 12.9

Consider the `BankAccountTester` class of Chapter 3. Which classes does it depend on?

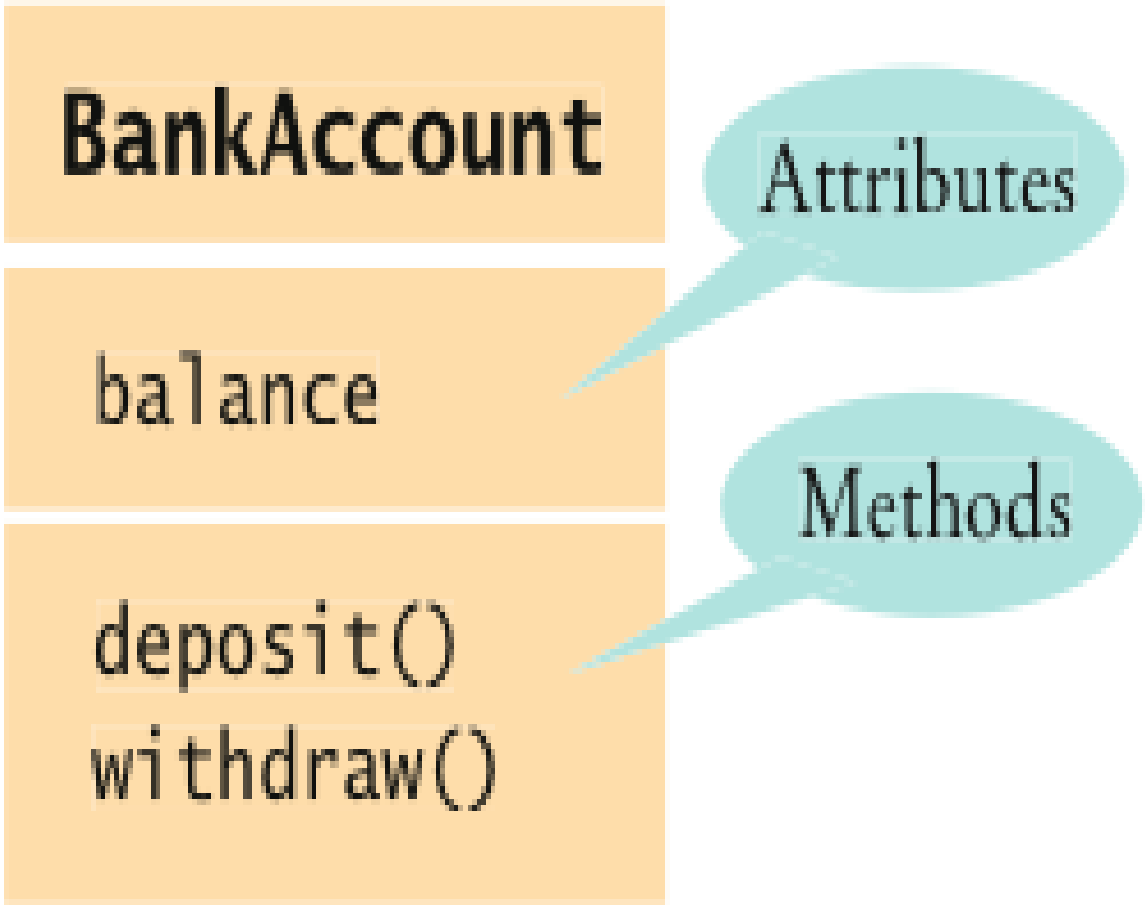
Answer: The `BankAccount`, `System`, and `PrintStream` classes.

- دروس أنها لا تعتمد على؟
- الجواب: إن BankAccount، نظام، والطبقات .PrintStream

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Attributes and Methods in UML Diagrams

سمات وأساليب في UMLرسم تخطيطي



Attributes and Methods in a Class Diagram

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

تجميع وجمعية

- 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

- جمعية: عن العلاقة العامة بين الطبقات
- استخدام في وقت مبكر من مرحلة التصميم
- ويرتبط فئة مع أخرى إذا يمكنك التنقل من وجوه فئة واحدة إلى كائنات أخرى
- نظرا كائن البنك، يمكنك التنقل إلى كائنات العملاء

An Association Relationship

Five-Part Development Process

خمسة اجزء عملية التطوير

1. Gather requirements
2. Use CRC cards to find classes, responsibilities, and collaborators
3. Use UML diagrams to record class relationships
4. Use `javadoc` to document method behavior
5. Implement your program

- جمع المتطلبات
- استخدام بطاقات CRCللعثور على الطبقات، والمسؤوليات، والمتعاونين
- استخدام مخططات UMLلتسجيل علاقات الطبقة
- استخدام `javadoc`لتوثيق طريقة السلوك
- تنفيذ البرنامج

Case Study: Printing an Invoice — Requirements

دراسة حالة: طباعة فاتورة - متطلبات

- Task: Print out an invoice
- Invoice: Describes the charges for a set of products in certain quantities
- Omit complexities
  - *Dates, taxes, and invoice and customer numbers*
- Print invoice
  - *Billing address, all line items, amount due*
- Line item
  - *Description, unit price, quantity ordered, total price*
- For simplicity, do not provide a user interface
- Test program: Adds line items to the invoice and then prints it

- المهمة: قم بطباعة فاتورة
- فاتورة: يصف الاتهامات لمجموعة من المنتجات بكميات معينة
- حذف التعقيدات
- مواعيد والضرائب والفواتير والعملاء أرقام
- طباعة الفاتورة
- عنوان الفواتير، وجميع البنود، المبلغ المستحق
- بند
- الوصف وسعر الوحدة، والكمية المطلوبة، السعر الإجمالي
- للبساطة، لا توفر واجهة المستخدم
- برنامج اختبار: إضافة بنود إلى الفاتورة ثم يطبع عليه

Case Study: Sample Invoice

دراسة حالة: نموذج الفاتورة

I N V O I C E

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

Description	Price	Qty	Total
Toaster	29.95	3	89.85
Hair dryer	24.95	1	24.95
Car vacuum	19.99	2	39.98

AMOUNT DUE: $154.78

Case Study: Printing an Invoice — CRC Cards

- Discover classes
- Nouns are possible classes:

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

- **دراسة حالة: طباعة فاتورة - بطاقات CRC**
- اكتشاف الطبقات
- الأسماء هي دروس ممكنة:
- فاتورة
- عنوان
- خط البند
- المنتج
- الوصف
- السعر
- الكمية
- إجمالي
- المبلغ المستحق

Case Study: Printing an Invoice — CRC Cards

- Analyze classes:

تحليل فئات هي:

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

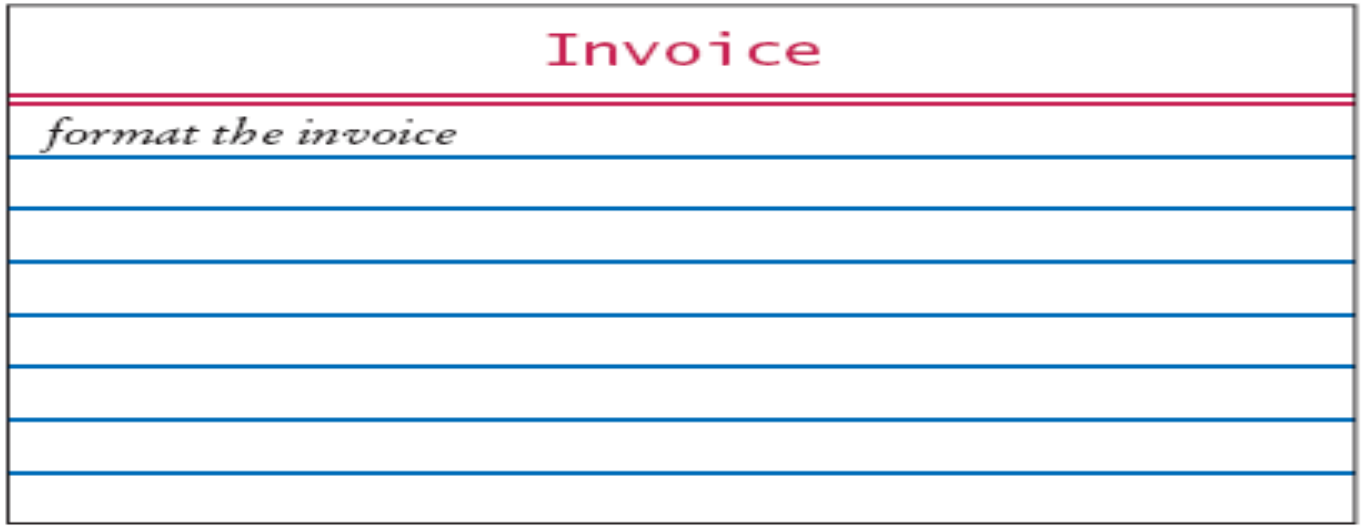
- Classes after a process of elimination:

فصول بعد عملية الإزالة:

```
Invoice
Address
LineItem
Product
```

CRC Cards for Printing Invoice

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

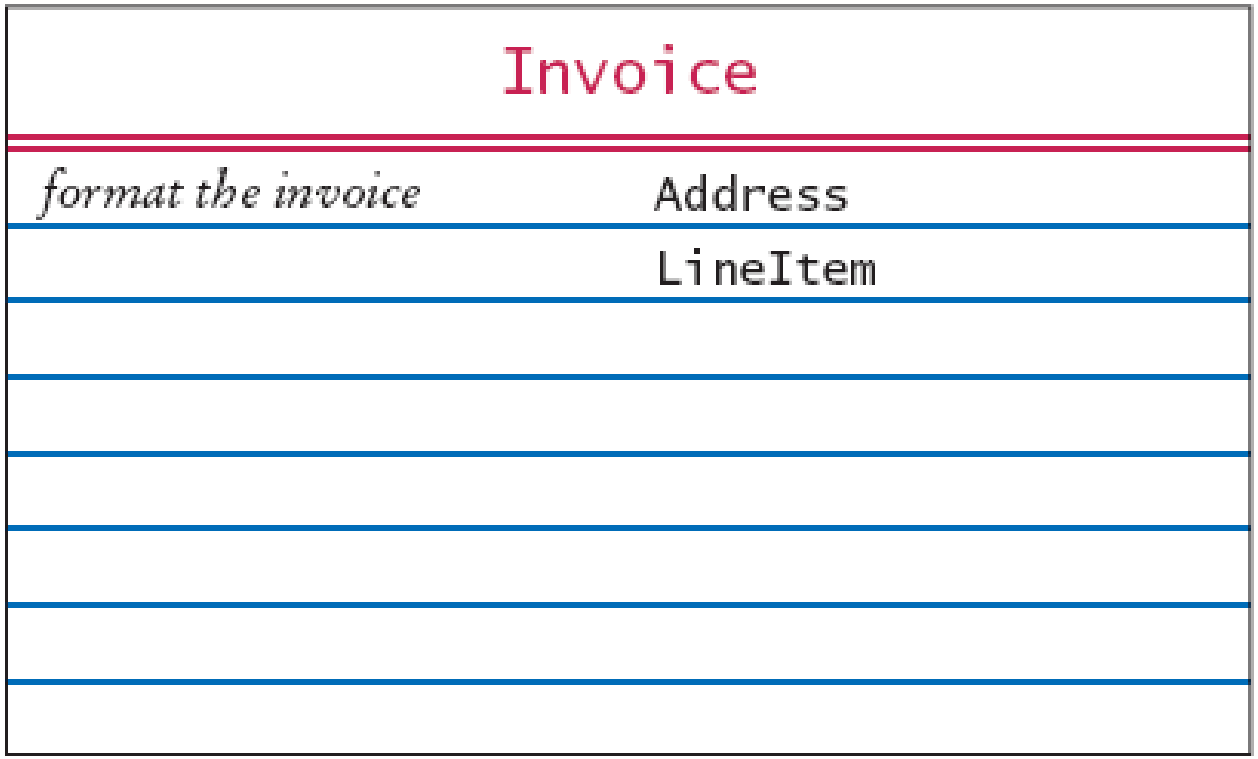


الفاتورة ويجب أن تكون قادرة على تهيئة أنفسهم العنوان:

CRC Cards for Printing Invoice

Add collaborators to invoice card:

إضافة المتعاونين لفاتورة بطاقة:



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:

يجب أن يتم ملؤها الفاتورة مع منتجات وكميات:

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

Printing an Invoice — UML Diagrams

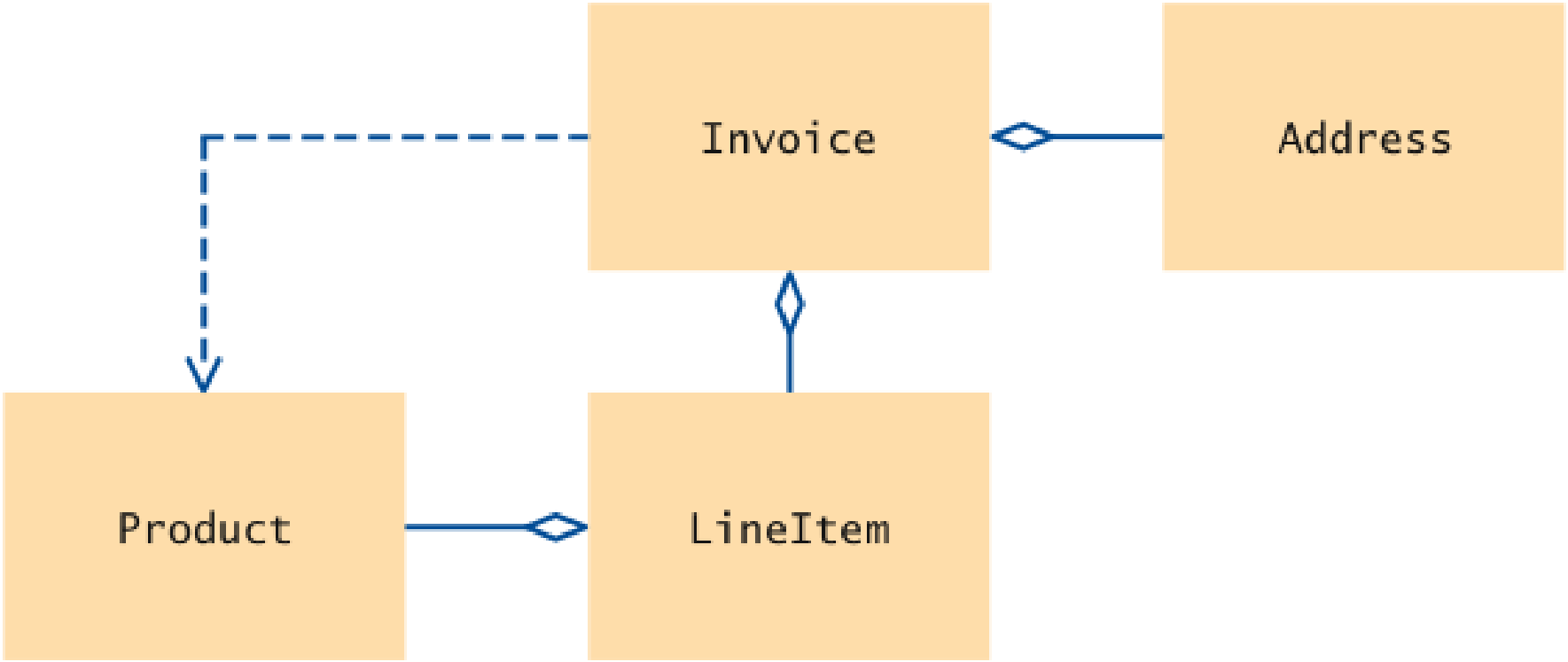


Figure 7 The Relationships Between the Invoice Classes

Printing an Invoice — Method Documentation

- Use `javadoc` documentation to record the behavior of the classes
- Leave the body of the methods blank
- Run `javadoc` to obtain formatted version of documentation in HTML format
- Advantages:
  - *Share HTML documentation with other team members*
  - *Format is immediately useful: Java source files*
  - *Supply the comments of the key methods*

- **طباعة فاتورة - طريقة التوثيق**

- استخدام وثائق في javadoc لتسجيل سلوك الطبقات
- ترك جثة الطرق فارغة
- تشغيل في javadoc للحصول على تنسيق إصدار الوثائق في شكل HTML
- المزايا:
- الوثائق حصة HTMLمع أعضاء الفريق الآخرين
- الشكل هو مفيد على الفور: الملفات المصدر جافا
- توريد تعليقات الطرق الرئيسية

Method Documentation — `Invoice` Class

طريقة التوثيق - فاتورة الفئة

```
/**
   Describes an invoice for a set of purchased products.
*/
public class Invoice
{
   /**
      Adds a charge for a product to this invoice.
      @param aProduct the product that the customer
         ordered
      @param quantity the quantity of the product
   */
   public void add(Product aProduct, int quantity)
   {
   }
```

Continued

Method Documentation — `Invoice` Class (cont.) (تابع)

```
   /**
      Formats the invoice.
      @return the formatted invoice
   */
   public String format()
   {
   }
}
```

Method Documentation – `LineItem` Class - الفئة LineItemطريقة التوثيق

```
/**
   Describes a quantity of an article to purchase and its
   price.
*/
public class LineItem
{
   /**
      Computes the total cost of this line item.
      @return the total price
   */
   public double getTotalPrice()
   {
   }
```

Continued

Method Documentation — `LineItem` Class (cont.) (تابع)

```
   /**
      Formats this item.
      @return a formatted string of this line item
   */
   public String format()
   {
   }
}
```

Method Documentation — `Product` Class

طريقة التوثيق - فئة المنتج

```
/**
   Describes a product with a description and a price.
*/
   public class Product
{
   /**
      Gets the product description.
      @return the description
   */
   public String getDescription()
   {
   }
```

Continued

Method Documentation — `Product` Class (cont.) (تابع)

```
   /**
      Gets the product price.
      @return the unit price
   */
   public double getPrice()
   {
   }
}
```

Method Documentation — `Address` Class طريقة التوثيق - الدرجة العنوان

```
/**
Describes a mailing address.
*/
public class Address
{
   /**
      Formats the address.
      @return the address as a string with three lines
   */
   public String format()
   {
   }
}
```

The Class Documentation in the HTML Format

وثائق فئة في تنسيق HTML

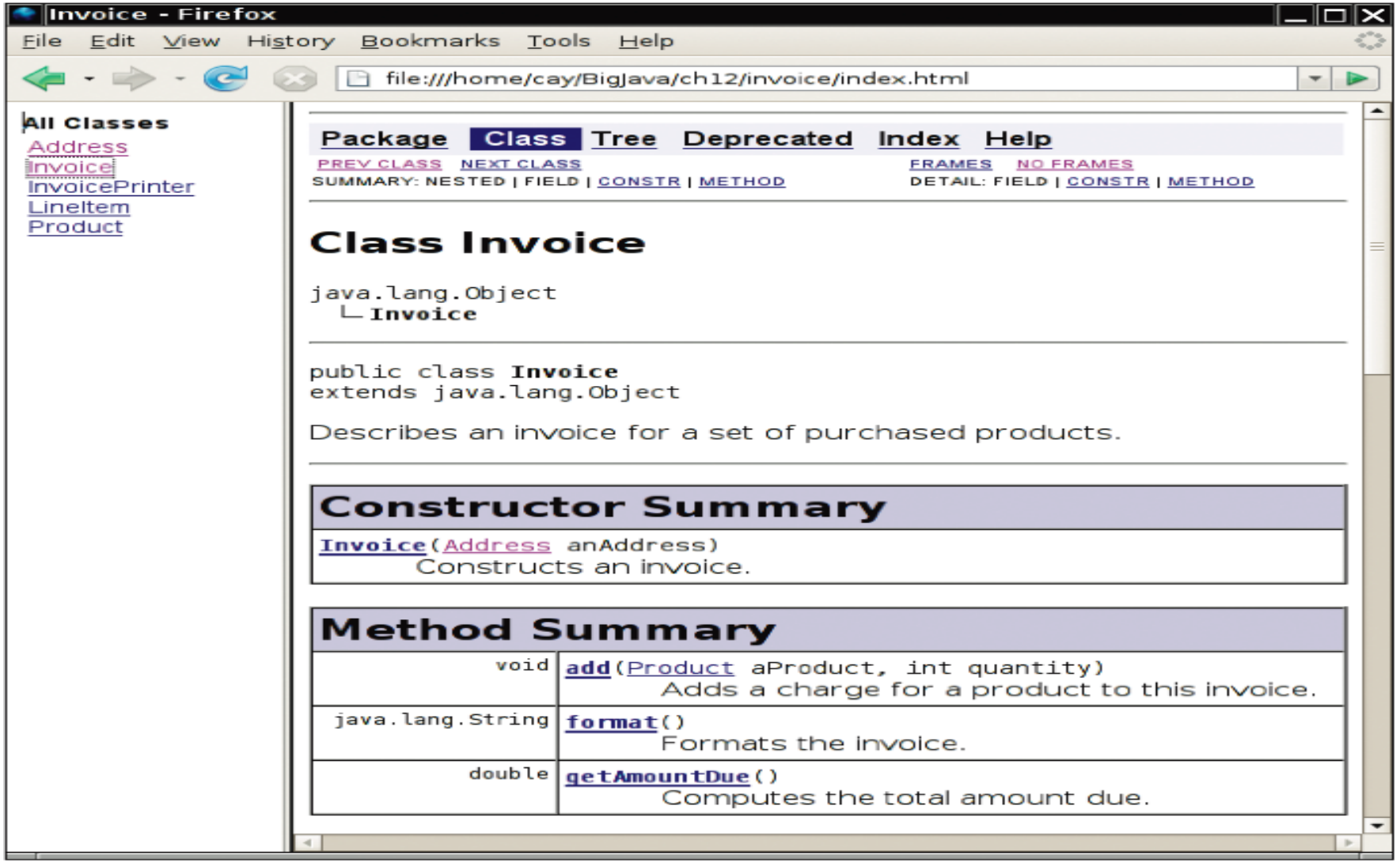


Figure 8 The Class Documentation in HTML Format

Printing an Invoice — Implementation

طباعة فاتورة - تنفيذ

- The UML diagram will give instance variables
- Look for associated classes
  - *They yield instance variables*

- والرسم البياني UMLتعطي المتغيرات المثال
- بحث عن الطبقات المرتبطة بها
- أنها تسفر عن المتغيرات المثال

Implementation

تطبيق

- `Invoice` aggregates `Address` and `LineItem`
- Every invoice has one billing address
- An invoice can have many line items:

```
public class Invoice
{
   ...
   private Address billingAddress;
   private ArrayList<LineItem> items;
}
```

- `Invoice` المجاميع `Address` و `LineItem`
- كل فاتورة لديه عنوان الفواتير واحد
- فاتورة يمكن أن يكون العديد من البنود:

Implementation

A line item needs to store a `Product` object and quantity:

```
public class LineItem
{
   ...
   private int quantity;
   private Product theProduct;
}
```

- A بند يحتاج إلى تخزين كائن المنتج والكمية:

Implementation

- The methods themselves are now very easy
- Example:
  - *`getTotalPrice` of `LineItem` gets the unit price of the product and multiplies it with the quantity:*

```
/**
   Computes the total cost of this line
item.
   @return the total price
*/
public double getTotalPrice()
{
   return theProduct.getPrice() *
quantity;
}
```

- الأساليب نفسها هي الآن من السهل جدا
- على سبيل المثال:
- getTotalPriceمن LineItem يحصل على سعر الوحدة من المنتج ويضاعف ذلك مع كمية:

ch12/invoice/InvoicePrinter.java

```
1  /**
2     This program demonstrates the invoice classes by printing
3     a sample invoice.
4  */
5  public class InvoicePrinter
6  {
7     public static void main(String[] args)
8     {
9        Address samsAddress
10             = new Address("Sam&apos;s Small Appliances",
11                "100 Main Street", "Anytown", "CA", "98765");
12
13       Invoice samsInvoice = new Invoice(samsAddress);
14       samsInvoice.add(new Product("Toaster", 29.95), 3);
15       samsInvoice.add(new Product("Hair dryer", 24.95), 1);
16       samsInvoice.add(new Product("Car vacuum", 19.99), 2);
17
18       System.out.println(samsInvoice.format());
19    }
20 }
21
22
23
```

ch12/invoice/Invoice.java

```
 1  import java.util.ArrayList;
 2
 3  /**
 4     Describes an invoice for a set of purchased products.
 5  */
 6  public class Invoice
 7  {
 8     private Address billingAddress;
 9     private ArrayList<LineItem> items;
10
11     /**
12        Constructs an invoice.
13        @param anAddress the billing address
14     */
15     public Invoice(Address anAddress)
16     {
17        items = new ArrayList<LineItem>();
18        billingAddress = anAddress;
19     }
20
```

Continued

ch12/invoice/Invoice.java (cont.)

```
21      /**
22         Adds a charge for a product to this invoice.
23         @param aProduct the product that the customer ordered
24         @param quantity the quantity of the product
25      */
26      public void add(Product aProduct, int quantity)
27      {
28         LineItem anItem = new LineItem(aProduct, quantity);
29         items.add(anItem);
30      }
31
```

Continued

ch12/invoice/Invoice.java (cont.)

```
32      /**
33         Formats the invoice.
34         @return the formatted invoice
35      */
36      public String format()
37      {
38         String r =  "                          I N V O I C E\n\n"
39               + billingAddress.format()
40               + String.format("\n\n%-30s%8s%5s%8s\n",
41                  "Description", "Price", "Qty", "Total");
42
43         for (LineItem item : items)
44         {
45            r = r + item.format() + "\n";
46         }
47
48         r = r + String.format("\nAMOUNT DUE: $%8.2f", getAmountDue());
49
50         return r;
51      }
52
```

ch12/invoice/Invoice.java (cont.)

```
53      /**
54         Computes the total amount due.
55         @return the amount due
56      */
57      public double getAmountDue()
58      {
59         double amountDue = 0;
60         for (LineItem item : items)
61         {
62            amountDue = amountDue + item.getTotalPrice();
63         }
64         return amountDue;
65      }
66   }
```

ch12/invoice/LineItem.java

```
1  /**
2     Describes a quantity of an article to purchase.
3  */
4  public class LineItem
5  {
6     private int quantity;
7     private Product theProduct;
8
9     /**
10       Constructs an item from the product and quantity.
11       @param aProduct the product
12       @param aQuantity the item quantity
13    */
14    public LineItem(Product aProduct, int aQuantity)
15    {
16       theProduct = aProduct;
17       quantity = aQuantity;
18    }
19
```

Continued

ch12/invoice/LineItem.java (cont.)

```
20      /**
21         Computes the total cost of this line item.
22         @return the total price
23      */
24      public double getTotalPrice()
25      {
26         return theProduct.getPrice() * quantity;
27      }
28
29      /**
30         Formats this item.
31         @return a formatted string of this item
32      */
33      public String format()
34      {
35         return String.format("%-30s%8.2f%5d%8.2f",
36            theProduct.getDescription(), theProduct.getPrice(),
37            quantity, getTotalPrice());
38      }
39   }
```

ch12/invoice/Product.java

```
1  /**
2     Describes a product with a description and a price.
3  */
4  public class Product
5  {
6     private String description;
7     private double price;
8
9     /**
10       Constructs a product from a description and a price.
11       @param aDescription the product description
12       @param aPrice the product price
13    */
14    public Product(String aDescription, double aPrice)
15    {
16       description = aDescription;
17       price = aPrice;
18    }
19
```

Continued

ch12/invoice/Product.java (cont.)

```
20      /**
21         Gets the product description.
22         @return the description
23      */
24      public String getDescription()
25      {
26         return description;
27      }
28
29      /**
30         Gets the product price.
31         @return the unit price
32      */
33      public double getPrice()
34      {
35         return price;
36      }
37   }
38
```

ch12/invoice/Address.java

```
1  /**
2     Describes a mailing address.
3  */
4  public class Address
5  {
6     private String name;
7     private String street;
8     private String city;
9     private String state;
10    private String zip;
11
12    /**
13       Constructs a mailing address.
14       @param aName the recipient name
15       @param aStreet the street
16       @param aCity the city
17       @param aState the two-letter state code
18       @param aZip the ZIP postal code
19    */
```

Continued

ch12/invoice/Address.java (cont.)

```
20      public Address(String aName, String aStreet,
21            String aCity, String aState, String aZip)
22      {
23         name = aName;
24         street = aStreet;
25         city = aCity;
26         state = aState;
27         zip = aZip;
28      }
29
30      /**
31         Formats the address.
32         @return the address as a string with three lines
33      */
34      public String format()
35      {
36         return name + "\n" + street + "\n"
37               + city + ", " + state + " " + zip;
38      }
39   }
40
```

Self Check 12.10

Which class is responsible for computing the amount due? What are its collaborators for this task?

Answer: The `Invoice` class is responsible for computing the amount due. It collaborates with the `LineItem` class.

- الفئة التي هي المسؤولة عن احتساب المبلغ المستحق؟ ما هي المتعاونين معها لهذه المهمة؟
- الجواب: فئة الفاتورة هي المسؤولة عن احتساب المبلغ المستحق. وتتعاون مع الطبقة .LineItem

Self Check 12.11

Why do the format methods return `String` objects instead of directly printing to `System.out`?

Answer: This design decision reduces coupling. It enables us to reuse the classes when we want to show the invoice in a dialog box or on a web page.

- لماذا الأساليب شكل تعود سلسلة الكائنات بدلا من الطباعة مباشرة إلى System.out؟
- الجواب: هذا القرار تصميم يقلل اقتران. إنها تمكننا من إعادة استخدام الطبقات عندما نريد أن تظهر الفاتورة في مربع الحوار أو على صفحة ويب.

Case Study: An Automatic Teller Machine — Requirements

- **دراسة حالة: آلة الصراف الآلي - متطلبات**

- ATM is used by bank customers. A customer has a
  - *Checking account*
  - *Savings account*
  - *Customer number*
  - *PIN*

- يستخدم الصراف الآلي من قبل عملاء البنك. عميل لديه
- حساب جار
- حساب التوفير
- رقم العميل
- PIN

Case Study: An Automatic Teller Machine — Requirements

- Customers can select an account
- The balance of the selected account is displayed
- Then, customer can deposit and withdraw money
- Process is repeated until the customer chooses to exit

- يمكن للعملاء اختيار حساب
- يتم عرض رصيد الحساب المحدد
- ثم، يمكن للعميل إيداع وسحب الأموال
- ويتكرر العملية حتى يختار العميل للخروج

Case Study: An Automatic Teller Machine — Requirements

- Two separate interfaces:
  - *GUI that closely mimics an actual ATM*
  - *Text-based interface*

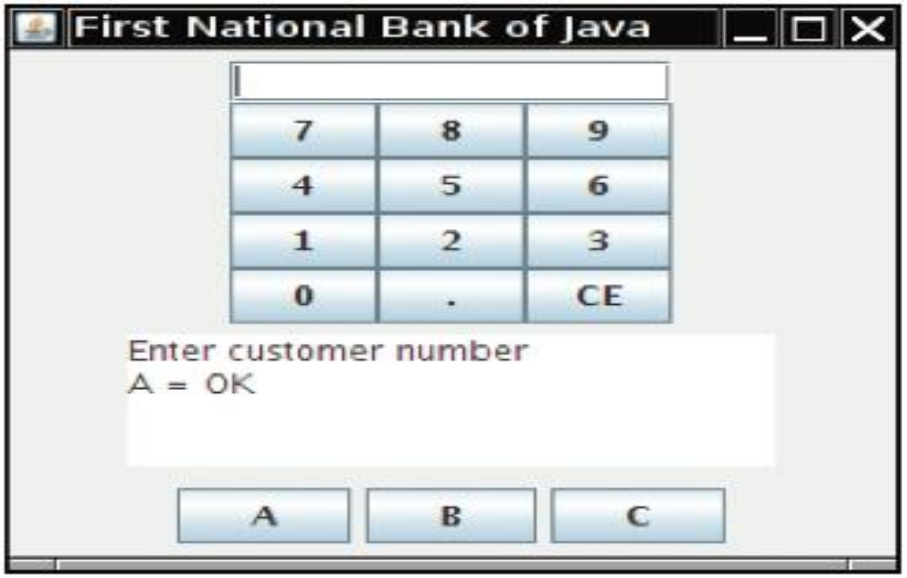
- واجهات اثنين منفصلة:
- GUIالذي يحاكي عن كثب أجهزة الصراف الآلي الفعلي
- واجهة يستند إلى نص

Case Study: An Automatic Teller Machine — Requirements

- GUI Interface
  - *Keypad*
  - *Display*
  - *Buttons A, B, C*
  - *Button functions depend on the state of the machine*

- واجهة المستخدم الرسومية
- لوحة المفاتيح
- عرض
- الأزرار A، B، C
- تعتمد وظائف زر عن حالة الجهاز

Figure 9
Graphical User Interface
for the Automatic Teller Machine



Case Study: An Automatic Teller Machine — Requirements

- At start up the customer is expected to
  - *Enter customer number*
  - *Press the A button*
  - *The display shows:*

```
Enter Customer Number
A = OK
```

- في البدء من المتوقع أن العملاء ل
- أدخل رقم العميل
- اضغط على زر A
- يظهر على الشاشة:

Case Study: An Automatic Teller Machine — Requirements

- The customer is expected to
  - *Enter a PIN*
  - *Press A button*
  - *The display shows:*

```
Enter PIN
A = OK
```

- ومن المتوقع أن العملاء ل
- أدخل PIN
- اضغط على زر
- يظهر على الشاشة:

Case Study: An Automatic Teller Machine — Requirements

- Search for the customer number and PIN
  - *If it matches a bank customer, proceed*
  - *Else return to start up screen*

- ابحث عن رقم العميل والرقم السري
- إذا كان يتطابق مع العملاء المصرفية، والمضي قدما
- عودة آخر لبدء التشغيل الشاشة

Case Study: An Automatic Teller Machine — Requirements

- If the customer is authorized
  - *The display shows:*

```
Select Account
A = Checking
B = Savings
C = Exit
```

- إذا أذن العملاء
- يظهر على الشاشة:

Case Study: An Automatic Teller Machine — Requirements

- If the user presses C
  - *The ATM reverts to its original state*
  - *ATM asks next user to enter a customer number*
- If the user presses A or B
  - *The ATM remembers selected account*
  - *The display shows:*

```
Balance = balance of selected account
Enter amount and select transaction
A = Withdraw
B = Deposit
C = Cancel
```

- إذا يضغط المستخدم C
- أجهزة الصراف الآلي يعود إلى حالته الأصلية
- ATMيسأل المستخدم التالي لإدخال رقم العميل
- إذا يضغط المستخدم A أو B
- أجهزة الصراف الآلي يتذكر الحساب المحدد
- يظهر على الشاشة:

Case Study: An Automatic Teller Machine — Requirements

- If the user presses A or B
  - *The value entered is withdrawn or deposited*
  - *Simulation: No money is dispensed and no deposit is accepted*
  - *The ATM reverts to previous state*
- If the user presses C
  - *The ATM reverts to previous state*

- إذا يضغط المستخدم Aأو B
- يتم سحب قيمة دخل أو إيداع
- المحاكاة: يتم الاستغناء لا مال ولا نتحمل الودائع
- أجهزة الصراف الآلي يعود إلى حالته السابقة
- إذا يضغط المستخدم C
- أجهزة الصراف الآلي يعود إلى حالته السابقة

Case Study: An Automatic Teller Machine — Requirements

- Text-based interaction
  - *Read input from* `System.in` *instead of the buttons*
  - *Here is a typical dialog:*

```
Enter account number: 1
Enter PIN: 1234
A=Checking, B=Savings, C=Quit: A
Balance=0.0
A=Deposit, B=Withdrawal, C=Cancel: A
Amount: 1000
A=Checking, B=Savings, C=Quit: C
```

- التفاعل القائم على النص
- قراءة المدخلات من System.in بدلا من الأزرار
- هنا هو الحوار نموذجي:

An Automatic Teller Machine – CRC Cards

CRCآلة الصراف الآلي - بطاقات

Nouns are possible classes:

الأسماء هي الطبقات محتملة

ATM	ATM
User	المستخدم
Keypad	لوحة المفاتيح
Display	عرض
Display message	رسالة العرض
Button	زر
State	حالة
Bank account	حساب البنك
Checking account	حساب جار
Savings account	حساب التوفير
Customer	العملاء
Customer number	رقم العميل
PIN	الرمز الشخصي
Bank	بنك

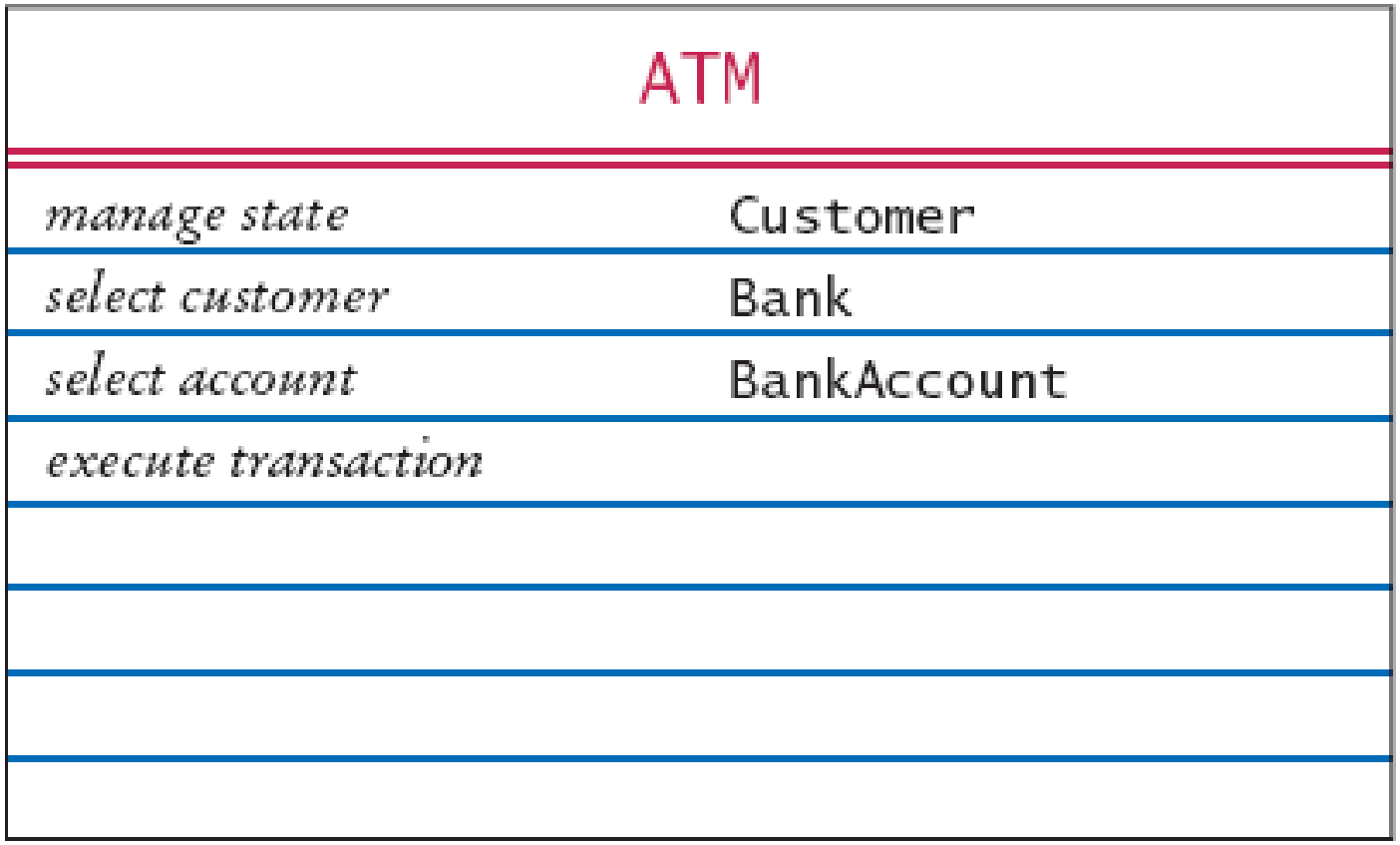
CRC Cards for Automatic Teller Machine

بطاقات CRC عن الآلية ATM

Customer	
get accounts	
match number and PIN	

Bank	
find customer	Customer
read customers	

CRC Cards for Automatic Teller Machine



ATM States

ATMالمتحدة

1. START: Enter customer ID
2. PIN: Enter PIN
3. ACCOUNT: Select account
4. TRANSACT: Select transaction

- البدء: أدخل IDالعملاء
- PIN: أدخل الرمز الشخصي
- ACCOUNT: حدد حساب
- TRANSACT: حدد المعاملات

State Diagram for ATM Class

الدولة مخطط ل ATMالفئة

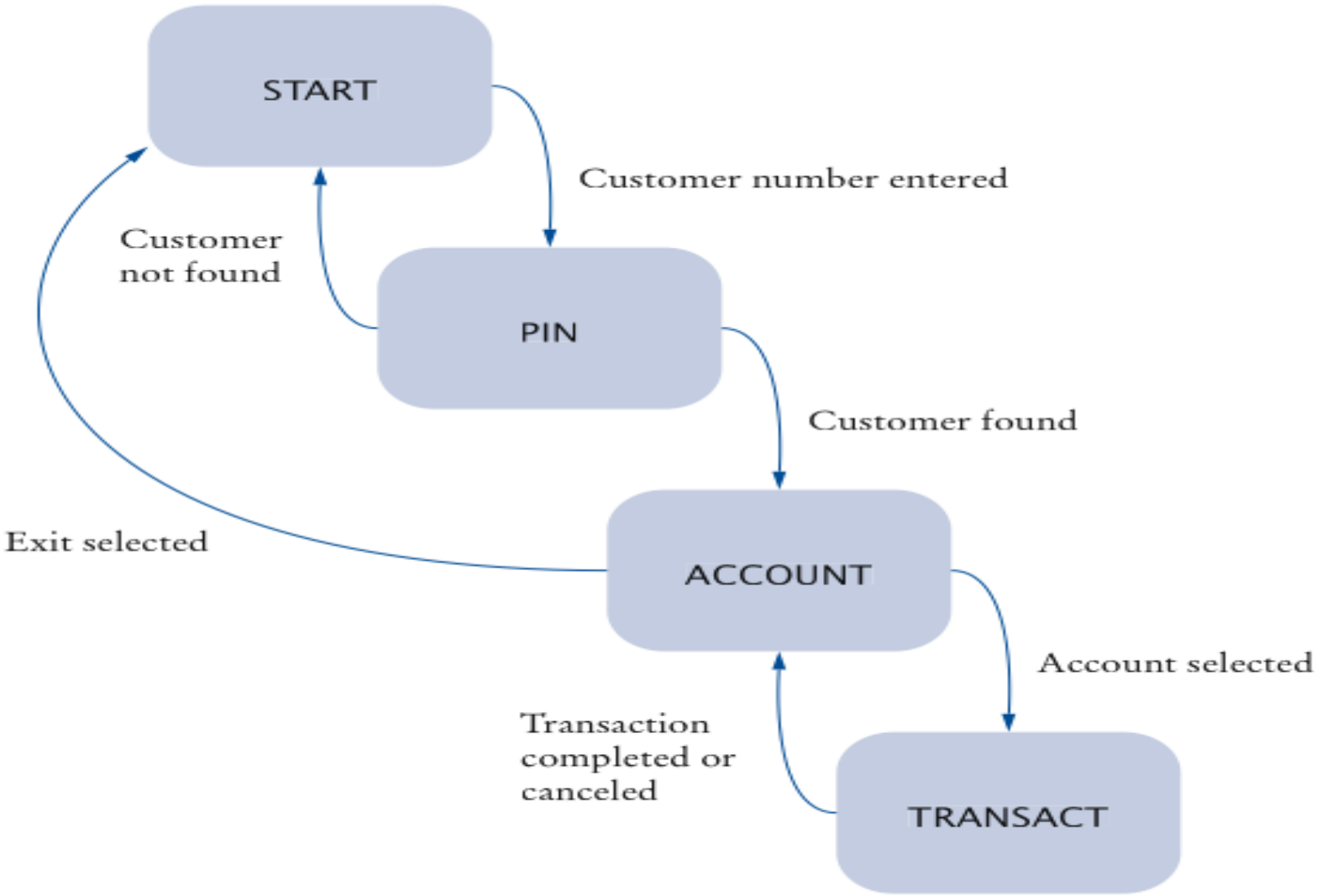


Figure 10 State Diagram for the ATM Class

An Automatic Teller Machine – UML Diagrams

رسم تخطيطي UMLآلة الصراف الآلي -

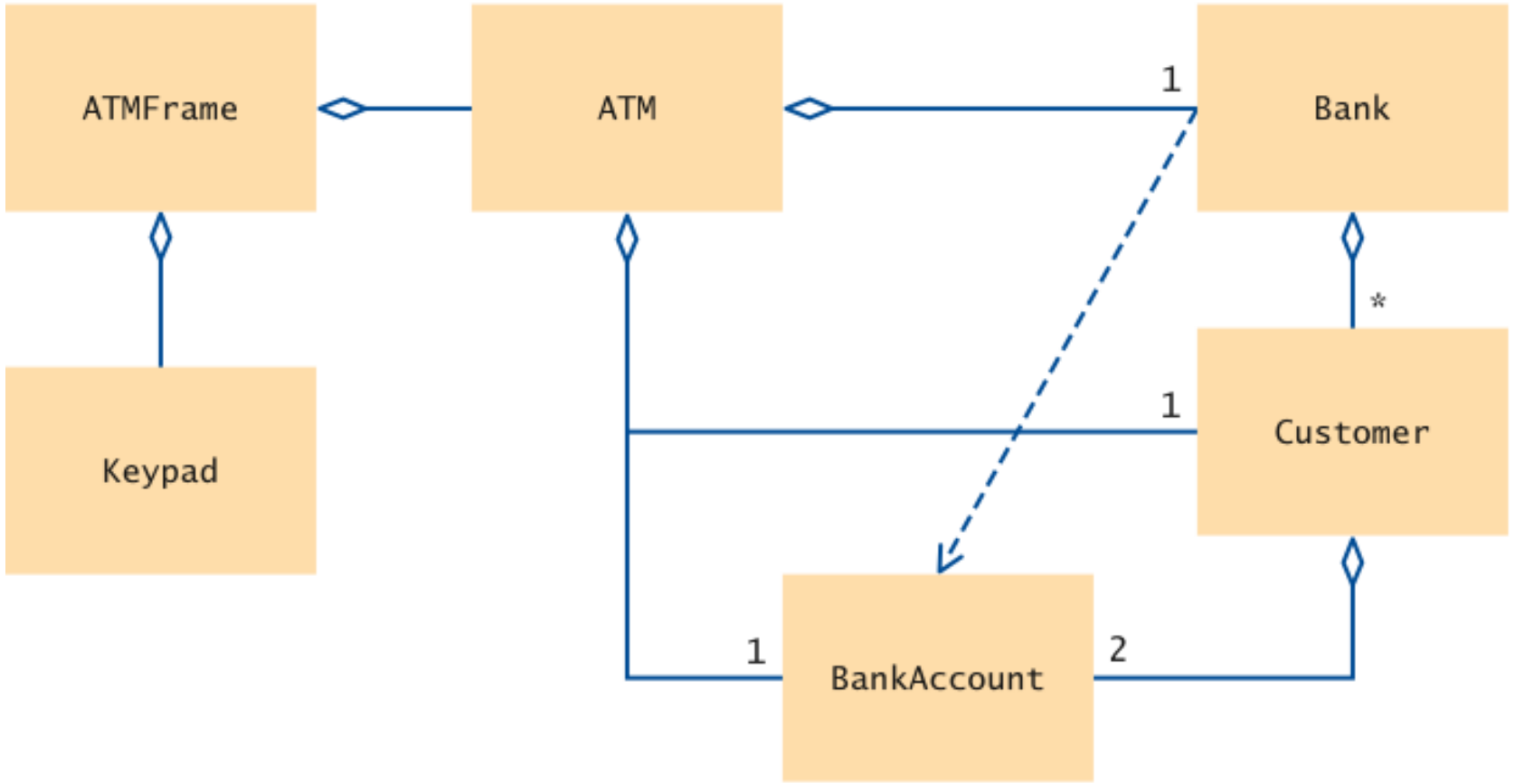


Figure 11 Relationships Between the ATM Classes

Method Documentation ATM Class

الفئة ATMطريقة التوثيق

```
/**
   An ATM that accesses a bank.
*/
public class ATM
{
   /**
      Constructs an ATM for a given bank.
      @param aBank the bank to which this ATM connects
   */
   public ATM(Bank aBank) { }
   /**
      Sets the current customer number
      and sets state to PIN.
      (Precondition: state is START)
      @param number the customer number
   */
   public void setCustomerNumber(int number) { }
```

Continued

Method Documentation ATM Class (cont.) (تابع)

```
/**
   Finds customer in bank.
   If found sets state to ACCOUNT, else to START.
   (Precondition: state is PIN)
   @param pin the PIN of the current customer
*/
public void selectCustomer(int pin) { }
/**
   Sets current account to checking or savings. Sets
   state to TRANSACT.
   (Precondition: state is ACCOUNT or TRANSACT)
   @param account one of CHECKING or SAVINGS
*/
```

Continued

Method Documentation ATM Class (cont.)

(تابع)

```
   public void selectAccount(int account) { }
   /**
      Withdraws amount from current account.
      (Precondition: state is TRANSACT)
      @param value the amount to withdraw
   */
   public void withdraw(double value) { }
   ...
}
```

An Automatic Teller Machine — Implementation

تنفيذ - ATMوالآلية

- Start implementation with classes that don't depend on others
  - `Keypad`
  - `BankAccount`
- Then implement `Customer` which depends only on `BankAccount`
- This bottom-up approach allows you to test your classes individually

- بدء تنفيذ مع الفئات التي لا تعتمد على الآخرين
- لوحة المفاتيح
- حساب البنك
- ثم تنفيذ العميل الذي يعتمد فقط على BankAccount
- هذا النهج التصاعدي يسمح لك لاختبار الطبقات بشكل فردي

An Automatic Teller Machine — Implementation

- Aggregated classes in UML diagram give instance variables

```
public class ATM
{
   private Bank theBank;
   ...
}
```

- الطبقات مجمعة في UMLالرسم البياني تعطي المتغيرات المثال

- From description of ATM states, it is clear that we require additional instance variables:

- من وصف الدول ATM، فمن الواضح أننا بحاجة المتغيرات المثال إضافية:

```
public class ATM
{
   private int state;
   private Customer currentCustomer;
   private BankAccount currentAccount;
   ...
}
```

An Automatic Teller Machine — Implementation

- Most methods are very straightforward to implement
- Consider `selectCustomer`:

- معظم الطرق هي واضحة جدا لتنفيذ
- النظر :selectCustomer

```
/**
   Finds customer in bank.
   If found sets state to ACCOUNT, else to START.
   (Precondition: state is PIN)
   @param pin the PIN of the current customer
*/
```

Continued

An Automatic Teller Machine — Implementation (cont.)

- Description can be almost literally translated to Java instructions:
- وصف يمكن ترجمتها حرفيا تقريبا لتعليمات جافا:

```
public void selectCustomer(int pin)
{
   assert state == PIN;
   currentCustomer = theBank.findCustomer(customerNumber,
         pin);
   if (currentCustomer == null)
      state = START;
   else
      state = ACCOUNT;
}
```

ch12/atm/ATM.java

```
1   /**
2      An ATM that accesses a bank.
3   */
4   public class ATM
5   {
6      public static final int CHECKING = 1;
7      public static final int SAVINGS = 2;
8
9      private int state;
10     private int customerNumber;
11     private Customer currentCustomer;
12     private BankAccount currentAccount;
13     private Bank theBank;
14
15     public static final int START = 1;
16     public static final int PIN = 2;
17     public static final int ACCOUNT = 3;
18     public static final int TRANSACT = 4;
19
```

ch12/atm/ATM.java (cont.)

```
20      /**
21         Constructs an ATM for a given bank.
22         @param aBank the bank to which this ATM connects
23      */
24      public ATM(Bank aBank)
25      {
26         theBank = aBank;
27         reset();
28      }
29
30      /**
31         Resets the ATM to the initial state.
32      */
33      public void reset()
34      {
35         customerNumber = -1;
36         currentAccount = null;
37         state = START;
38      }
39
```

ch12/atm/ATM.java (cont.)

```
40      /**
41         Sets the current customer number
42         and sets state to PIN.
43         (Precondition: state is START)
44         @param number the customer number.
45      */
46      public void setCustomerNumber(int number)
47      {
48         assert state == START;
49         customerNumber = number;
50         state = PIN;
51      }
52
```

ch12/atm/ATM.java (cont.)

```
53      /**
54         Finds customer in bank.
55         If found sets state to ACCOUNT, else to START.
56         (Precondition: state is PIN)
57         @param pin the PIN of the current customer
58      */
59      public void selectCustomer(int pin)
60      {
61         assert state == PIN;
62     currentCustomer =
theBank.findCustomer(customerNumber, pin);
63         if (currentCustomer == null)
64            state = START;
65         else
66            state = ACCOUNT;
67      }
68
```

ch12/atm/ATM.java (cont.)

```
69      /**
70         Sets current account to checking or savings. Sets
71         state to TRANSACT.
72         (Precondition: state is ACCOUNT or TRANSACT)
73         @param account one of CHECKING or SAVINGS
74      */
75      public void selectAccount(int account)
76      {
77         assert state == ACCOUNT || state == TRANSACT;
78         if (account == CHECKING)
79            currentAccount =
currentCustomer.getCheckingAccount();
80         else
81            currentAccount =
currentCustomer.getSavingsAccount();
82         state = TRANSACT;
83      }
84
```

ch12/atm/ATM.java (cont.)

```
85      /**
86         Withdraws amount from current account.
87         (Precondition: state is TRANSACT)
88         @param value the amount to withdraw
89      */
90      public void withdraw(double value)
91      {
92         assert state == TRANSACT;
93         currentAccount.withdraw(value);
94      }
95
96      /**
97         Deposits amount to current account.
98         (Precondition: state is TRANSACT)
99         @param value the amount to deposit
100     */
101     public void deposit(double value)
102     {
103        assert state == TRANSACT;
104        currentAccount.deposit(value);
105     }
106
```

ch12/atm/ATM.java (cont.)

```
107      /**
108         Gets the balance of the current account.
109         (Precondition: state is TRANSACT)
110         @return the balance
111      */
112      public double getBalance()
113      {
114         assert state == TRANSACT;
115         return currentAccount.getBalance();
116      }
117
118      /**
119         Moves back to the previous state.
120      */
121      public void back()
122      {
123         if (state == TRANSACT)
124            state = ACCOUNT;
125         else if (state == ACCOUNT)
126            state = PIN;
127         else if (state == PIN)
128            state = START;
129      }
130
```

ch12/atm/ATM.java (cont.)

```
131       /**
132          Gets the current state of this ATM.
133          @return the current state
134       */
135       public int getState()
136       {
137          return state;
138       }
139    }
```

ch12/atm/Bank.java

```
1  import java.io.File;
2  import java.io.IOException;
3  import java.util.ArrayList;
4  import java.util.Scanner;
5
6  /**
7     A bank contains customers with bank accounts.
8  */
9  public class Bank
10 {
11    private ArrayList<Customer> customers;
12
13    /**
14       Constructs a bank with no customers.
15    */
16    public Bank()
17    {
18       customers = new ArrayList<Customer>();
19    }
20
```

Continued

ch12/atm/Bank.java (cont.)

```
21      /**
22         Reads the customer numbers and pins
23         and initializes the bank accounts.
24         @param filename the name of the customer file
25      */
26      public void readCustomers(String filename)
27            throws IOException
28      {
29         Scanner in = new Scanner(new File(filename));
30         while (in.hasNext())
31         {
32            int number = in.nextInt();
33            int pin = in.nextInt();
34            Customer c = new Customer(number, pin);
35            addCustomer(c);
36         }
37         in.close();
38      }
39
```

Continued

ch12/atm/Bank.java (cont.)

```
40      /**
41         Adds a customer to the bank.
42         @param c the customer to add
43      */
44      public void addCustomer(Customer c)
45      {
46         customers.add(c);
47      }
48
49      /**
50         Finds a customer in the bank.
51         @param aNumber a customer number
52         @param aPin a personal identification number
53         @return the matching customer, or null if no customer
54         matches
55      */
56      public Customer findCustomer(int aNumber, int aPin)
57      {
58         for (Customer c : customers)
59         {
60            if (c.match(aNumber, aPin))
61               return c;
62         }
63         return null;
64      }
65   }
```

ch12/atm/Customer.java

```
1  /**
2     A bank customer with a checking and a savings account.
3  */
4  public class Customer
5  {
6     private int customerNumber;
7     private int pin;
8     private BankAccount checkingAccount;
9     private BankAccount savingsAccount;
10
11    /**
12       Constructs a customer with a given number and PIN.
13       @param aNumber the customer number
14       @param aPin the personal identification number
15    */
16    public Customer(int aNumber, int aPin)
17    {
18       customerNumber = aNumber;
19       pin = aPin;
20       checkingAccount = new BankAccount();
21       savingsAccount = new BankAccount();
22    }
23
```

Continued

ch12/atm/Customer.java (cont.)

```
24      /**
25         Tests if this customer matches a customer number
26         and PIN.
27         @param aNumber a customer number
28         @param aPin a personal identification number
29         @return true if the customer number and PIN match
30      */
31      public boolean match(int aNumber, int aPin)
32      {
33         return customerNumber == aNumber && pin == aPin;
34      }
35
36      /**
37         Gets the checking account of this customer.
38         @return the checking account
39      */
40      public BankAccount getCheckingAccount()
41      {
42         return checkingAccount;
43      }
44
```

Continued

ch12/atm/Customer.java (cont.)

```
45      /**
46         Gets the savings account of this customer.
47         @return the checking account
48      */
49      public BankAccount getSavingsAccount()
50      {
51         return savingsAccount;
52      }
53   }
```

ch12/atm/ATMSimulator.java

```
1  import java.io.IOException;
2  import java.util.Scanner;
3
4  /**
5     A text-based simulation of an automatic teller machine.
6  */
7  public class ATMSimulator
8  {
9     public static void main(String[] args)
10    {
11       ATM theATM;
12       try
13       {
14          Bank theBank = new Bank();
15          theBank.readCustomers("customers.txt");
16          theATM = new ATM(theBank);
17       }
18       catch(IOException e)
19       {
20          System.out.println("Error opening accounts file.");
21          return;
22       }
23
```

Continued

ch12/atm/ATMSimulator.java (cont.)

```
24          Scanner in = new Scanner(System.in);
25
26          while (true)
27          {
28             int state = theATM.getState();
29             if (state == ATM.START)
30             {
31                System.out.print("Enter customer number:
");
32                int number = in.nextInt();
33                theATM.setCustomerNumber(number);
34             }
35             else if (state == ATM.PIN)
36             {
37                System.out.print("Enter PIN: ");
38                int pin = in.nextInt();
39                theATM.selectCustomer(pin);
40             }
```

ch12/atm/ATMSimulator.java (cont.)

```
41            else if (state == ATM.ACCOUNT)
42            {
43               System.out.print("A=Checking, B=Savings, C=Quit:
");
44               String command = in.next();
45               if (command.equalsIgnoreCase("A"))
46                  theATM.selectAccount(ATM.CHECKING);
47               else if (command.equalsIgnoreCase("B"))
48                  theATM.selectAccount(ATM.SAVINGS);
49               else if (command.equalsIgnoreCase("C"))
50                  theATM.reset();
51               else
52                  System.out.println("Illegal input!");
53            }
54            else if (state == ATM.TRANSACT)
55            {
56               System.out.println("Balance=" +
theATM.getBalance());
57               System.out.print("A=Deposit, B=Withdrawal,
C=Cancel: ");
58               String command = in.next();
```

ch12/atm/ATMSimulator.java (cont.)

```
59                  if (command.equalsIgnoreCase("A"))
60                  {
61                     System.out.print("Amount: ");
62                     double amount = in.nextDouble();
63                     theATM.deposit(amount);
64                     theATM.back();
65                  }
66                  else if (command.equalsIgnoreCase("B"))
67                  {
68                     System.out.print("Amount: ");
69                     double amount = in.nextDouble();
70                     theATM.withdraw(amount);
71                     theATM.back();
72                  }
73                  else if (command.equalsIgnoreCase("C"))
74                     theATM.back();
75                  else
76                     System.out.println("Illegal input!");
77               }
78            }
79         }
80   }
```

ch12/atm/ATMSimulator.java (cont.)

Program Run:

```
Enter account number: 1
Enter PIN: 1234
A=Checking, B=Savings, C=Quit: A
Balance=0.0
A=Deposit, B=Withdrawal, C=Cancel: A
Amount: 1000
A=Checking, B=Savings, C=Quit: C
...
```

ch12/atm/ATMViewer.java

```
1  import java.io.IOException;
2  import javax.swing.JFrame;
3  import javax.swing.JOptionPane;
4
5  /**
6     A graphical simulation of an automatic teller machine.
7  */
8  public class ATMViewer
9  {
10    public static void main(String[] args)
11    {
12       ATM theATM;
13
14       try
15       {
16          Bank theBank = new Bank();
17          theBank.readCustomers("customers.txt");
18          theATM = new ATM(theBank);
19       }
20       catch(IOException e)
21       {
22          JOptionPane.showMessageDialog(null, "Error opening accounts
file.");
23          return;
24       }
25
```

ch12/atm/ATMViewer.java (cont.)

```
26          JFrame frame = new ATMFrame(theATM);
27          frame.setTitle("First National Bank of Java");
28          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29          frame.setVisible(true);
30       }
31    }
32
```

ch12/atm/ATMFrame.java

```
1  import java.awt.FlowLayout;
2  import java.awt.GridLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import javax.swing.JButton;
6  import javax.swing.JFrame;
7  import javax.swing.JPanel;
8  import javax.swing.JTextArea;
9
10 /**
11    A frame displaying the components of an ATM.
12 */
13 public class ATMFrame extends JFrame
14 {
15    private static final int FRAME_WIDTH = 300;
16    private static final int FRAME_HEIGHT = 300;
17
18    private JButton aButton;
19    private JButton bButton;
20    private JButton cButton;
21
22    private KeyPad pad;
23    private JTextArea display;
```

ch12/atm/ATMFrame.java (cont.)

```
24
25      private ATM theATM;
26
27      /**
28         Constructs the user interface of the ATM frame.
29      */
30      public ATMFrame(ATM anATM)
31      {
32         theATM = anATM;
33
34         // Construct components
35         pad = new KeyPad();
36
37         display = new JTextArea(4, 20);
38
39         aButton = new JButton("  A  ");
40         aButton.addActionListener(new AButtonListener());
41
42         bButton = new JButton("  B  ");
43         bButton.addActionListener(new BButtonListener());
44
```

ch12/atm/ATMFrame.java (cont.)

```
45          cButton = new JButton("  C  ");
46          cButton.addActionListener(new CButtonListener());
47
48          // Add components
49
50          JPanel buttonPanel = new JPanel();
51          buttonPanel.add(aButton);
52          buttonPanel.add(bButton);
53          buttonPanel.add(cButton);
54
55          setLayout(new FlowLayout());
56          add(pad);
57          add(display);
58          add(buttonPanel);
59          showState();
60
61          setSize(FRAME_WIDTH, FRAME_HEIGHT);
62      }
63
```

ch12/atm/ATMFrame.java (cont.)

```
64      /**
65         Updates display message.
66      */
67      public void showState()
68      {
69         int state = theATM.getState();
70         pad.clear();
71         if (state == ATM.START)
72            display.setText("Enter customer number\nA = OK");
73         else if (state == ATM.PIN)
74            display.setText("Enter PIN\nA = OK");
75         else if (state == ATM.ACCOUNT)
76            display.setText("Select Account\n"
77                  + "A = Checking\nB = Savings\nC = Exit");
78         else if (state == ATM.TRANSACT)
79            display.setText("Balance = "
80                  + theATM.getBalance()
81                  + "\nEnter amount and select transaction\n"
82                  + "A = Withdraw\nB = Deposit\nC = Cancel");
83      }
84
```

ch12/atm/ATMFrame.java (cont.)

```
85      class AButtonListener implements ActionListener
86      {
87         public void actionPerformed(ActionEvent event)
88         {
89            int state = theATM.getState();
90            if (state == ATM.START)
91               theATM.setCustomerNumber((int) pad.getValue());
92            else if (state == ATM.PIN)
93               theATM.selectCustomer((int) pad.getValue());
94            else if (state == ATM.ACCOUNT)
95               theATM.selectAccount(ATM.CHECKING);
96            else if (state == ATM.TRANSACT)
97            {
98               theATM.withdraw(pad.getValue());
99               theATM.back();
100           }
101           showState();
102        }
103     }
104
```

ch12/atm/ATMFrame.java (cont.)

```
105      class BButtonListener implements ActionListener
106      {
107         public void actionPerformed(ActionEvent event)
108         {
109            int state = theATM.getState();
110            if (state == ATM.ACCOUNT)
111               theATM.selectAccount(ATM.SAVINGS);
112            else if (state == ATM.TRANSACT)
113            {
114               theATM.deposit(pad.getValue());
115               theATM.back();
116            }
117            showState();
118         }
119      }
120
```

Continued

ch12/atm/ATMFrame.java (cont.)

```
121      class CButtonListener implements ActionListener
122      {
123         public void actionPerformed(ActionEvent event)
124         {
125            int state = theATM.getState();
126            if (state == ATM.ACCOUNT)
127               theATM.reset();
128            else if (state == ATM.TRANSACT)
129               theATM.back();
130            showState();
131         }
132      }
133   }
```

ch12/atm/KeyPad.java

```
1  import java.awt.BorderLayout;
2  import java.awt.GridLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import javax.swing.JButton;
6  import javax.swing.JPanel;
7  import javax.swing.JTextField;
8
9  /**
10    A component that lets the user enter a number, using
11    a button pad labeled with digits.
12 */
13 public class KeyPad extends JPanel
14 {
15    private JPanel buttonPanel;
16    private JButton clearButton;
17    private JTextField display;
18
```

Continued

ch12/atm/KeyPad.java (cont.)

```
36          // Add digit buttons
37
38          addButton("7");
39          addButton("8");
40          addButton("9");
41          addButton("4");
42          addButton("5");
43          addButton("6");
44          addButton("1");
45          addButton("2");
46          addButton("3");
47          addButton("0");
48          addButton(".");
49
50          // Add clear entry button
51
52          clearButton = new JButton("CE");
53          buttonPanel.add(clearButton);
54
```

ch12/atm/KeyPad.java (cont.)

```
55          class ClearButtonListener implements ActionListener
56          {
57             public void actionPerformed(ActionEvent event)
58             {
59                display.setText("");
60             }
61          }
62          ActionListener listener = new ClearButtonListener();
63
64          clearButton.addActionListener(new
65                ClearButtonListener());
66
67          add(buttonPanel, "Center");
68       }
69
```

ch12/atm/KeyPad.java (cont.)

```
70      /**
71         Adds a button to the button panel
72         @param label the button label
73      */
74      private void addButton(final String label)
75      {
76         class DigitButtonListener implements ActionListener
77         {
78            public void actionPerformed(ActionEvent event)
79            {
80
81               // Don&apos;t add two decimal points
82               if (label.equals(".")
83                     && display.getText().indexOf(".") != -1)
84                  return;
85
86               // Append label text to button
87               display.setText(display.getText() + label);
88            }
89         }
90
```

ch12/atm/KeyPad.java (cont.)

```
91          JButton button = new JButton(label);
92          buttonPanel.add(button);
93          ActionListener listener = new DigitButtonListener();
94          button.addActionListener(listener);
95       }
96
97       /**
98          Gets the value that the user entered.
99          @return the value in the text field of the keypad
100      */
101      public double getValue()
102      {
103         return Double.parseDouble(display.getText());
104      }
105
106      /**
107         Clears the display.
108      */
109      public void clear()
110      {
111         display.setText("");
112      }
113   }
```

Self Check 12.12

Why does the `Bank` class in this example not store an array list of bank accounts?

Answer: The bank needs to store the list of customers so that customers can log in. We need to locate all bank accounts of a customer, and we chose to simply store them in the customer class. In this program, there is no further need to access bank accounts.

- لماذا الطبقة البنك في هذا المثال تقم بتخزين قائمة مجموعة من الحسابات المصرفية؟
- الإجابة: يحتاج البنك لتخزين قائمة من العملاء بحيث يمكن للعملاء الدخول نحن بحاجة إلى العثور على جميع الحسابات المصرفية للعملاء، ولقد اخترنا لمجرد تخزينها في فئة العملاء. في هذا البرنامج، ليست هناك حاجة إضافية للوصول إلى حسابات مصرفية.

Self Check 12.13

Suppose the requirements change — you need to save the current account balances to a file after every transaction and reload them when the program starts. What is the impact of this change on the design?

Answer: The `Bank` class needs to have an additional responsibility: to load and save the accounts. The bank can carry out this responsibility because it has access to the customer objects and, through them, to the bank accounts.

- لنفترض تتغير متطلبات - تحتاج إلى حفظ أرصدة الحسابات الجارية إلى ملف بعد كل معاملة وإعادة تحميلها عند بدء تشغيل البرنامج. ما هو تأثير هذا التغيير على التصميم؟
- الإجابة: فئة البنك يحتاج الى مبلغ إضافي؟ المسؤولية: لتحميل وحفظ حسابات. ويمكن للبنك؟ تنفيذ هذه المسؤولية لأنه لديه حق الوصول إلى؟ كائنات العميل، ومن خلالهم، إلى حسابات مصرفية.