UML and OOD

- What are good systems like?
- Most real systems are too big for a single developer to understand everything about the system ⇒ modules.
- How do we break a system into modules/components/classes?

Some terms related to OOD

- Dependency
- Coupling
- Cohesion
- Interface
- Encapsulation
- Abstraction

Dependency

- Class A depends on class B if a change in B can necessitate a change in A.
- A is said to be a client of B.
- A circular dependence is bad because it hampers reuse. You can’t reuse one class in the circle without using all of them.

Coupling

- A system with many dependencies has high coupling (that’s bad).
- Low coupling implies changes in one part affect fewer other parts.
- How can you identify dependencies?

Interfaces

- An interface defines services exported from a class or module.
- The interface identifies the methods the clients depend on.
- Java checks syntactic dependencies (type checking and method existence).
- Java does not support semantic checking or enforcement of dependencies.
- For example, you cannot specify formally that a sort() method actually sorts.

Multiple Interfaces

- Implementing multiple interfaces allows for more accurate representation of dependencies.
- For example, by using the Painter interface, I documented that PenAdjuster depends on PaintListener3 have a setPenSize() method.
- Without this interface, it was not superficially obvious that PenAdjuster didn’t depend on other aspects of PaintListener3.
Cohesion

- A class/module with an intuitively simple interface that is in fact complex to implement has high **cohesion**.
- Its parts “belong” together.
- The interface provides an **abstraction** of the internally complex object.

Abstraction vs Encapsulation

- Abstraction: the client only *needs* to know the interface.
- Encapsulation: the client *cannot* see more than the interface.
- High cohesion and low coupling ⇒ easy to replace, easy to reuse, easy to understand/maintain/modify ⇒ Good!

UML: A Case Study

- You have been asked to build a university library computer system.
- We will use UML to work through part of the design.
- First we need to refine the requirements.

Requirement: Books and Journals

The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Only members of staff may borrow journals. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. New books and journals arrive regularly, and old ones are sometimes disposed of. The current year’s journals are sent away to be bound into volumes at the end of each year.

Requirement: Borrowing

It is essential that the system keeps track of when books and journals are borrowed and returned, since the current system already does that. The new system should produce reminders when a book is overdue. There may in the future be a requirement for users to be able to extend the loan of a book if it is not reserved.

Requirement: Browsing

The system should allow users to search for a book on a particular topic, by a particular author, etc., to check whether a copy of the book is available for loan and, if not, to reserve the book. Anybody can browse in the library.
Simplified 1st Version

Books and journals: The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

Borrowing: The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

Use Case Model

• Users – actors in UML (usually people), e.g. BookBorrower
• Tasks – use cases, e.g. Borrow copy of book
• A use case is a task which a user must perform with the help of the system – e.g. borrow a book.
• Write a detailed description, in third-person, active-voice, for each “use case.”

A Use Case

Borrow copy of book – A BookBorrower presents a book. The system checks that the potential borrower is a member of the library, and that s/he does not already have the maximum permitted number of books on loan. This maximum is six unless the member is a staff member, in which case it is 12. If both checks succeed, the system records that this library member has this copy of the book on loan. Otherwise it refuse the loan.

Use Case Diagram

Use Case Descriptions

• You must write use case descriptions for each use case shown in the use case diagram.
• The goal is to document what the system “should do” not “how” it should do it.
• Do not invent requirements while writing these descriptions.

Identifying Classes: The noun identification technique

Books and journals: The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

Borrowing: The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.
The Initial Set of Classes

• After eliminating the “obviously” poor candidates we are left with the following possible classes:
• Book
• Journal
• Copy (of book)
• Library member
• Member of staff

CRC Cards

• At this point you could create CRC cards for the classes and begin to identify some responsibilities.
• We can also begin to identify some collaborations.
• In UML collaborations are called associations.

Class Diagrams and Associations

• Class diagrams are the UML equivalent of CRC cards.
• Associations describe how objects work together – the “collaboration” in CRC.
• Associations allow us to check coupling.

A Class Diagram

- Note the multiplicity notations.
- Note the lack of direction (navigability) on associations.
- We are not concerned at this point about which class depends on which.

Class Diagrams

- Note the multiplicity notations.
- Note the lack of direction (navigability) on associations.
- We are not concerned at this point about which class depends on which.
Collaboration Diagrams

- Class diagrams show a static view of the system.
- Collaboration diagrams show a more dynamic view of the system in action.
- A collaboration diagram corresponds to a particular use case.
- Not all use cases will require collaboration diagrams.

Collaboration Diagram

- Note the numbering scheme for the messages.
- Messages imply associations and navigability in the class diagram.

Borrow copy of book

- \text{aMember:BookBorrower}
  - \text{Borrow (theCopy)}
  - 2.1: \text{borrowed}

- \text{theLibraryMember:LibraryMember}
  - 2.\text{borrow}
  - \text{1:okToBorrow}

- \text{theCopy:Copy}
  - \text{theBook:Book}