Slide 1

What is UML?

- "Unified Modeling Language".
- From originators' Web site:

  "method for specifying, visualizing, and documenting the artifacts of an object-oriented system under development"
- From *UML Distilled*:

  "family of graphical notations, backed by single meta-model, that help in describing and designing software systems, particularly software systems built using the object–oriented style"
- Many things to many people, used in different ways.

Slide 2

Why Model?

- Large and complex systems are difficult for humans to understand.
- Abstracting out key features ("modeling") and representing pictorially helps.

Slide 3

Why Use UML?

- Pictorial representations of models are often easier to understand than prose. Pictorial representations using a standard notation are even better.
- Prior to UML, much experimentation with different ways of modeling object-oriented systems. Goal of UML was to unify these.
- A key point — all elements of UML are easy to draw, allow representing ideas at different levels of detail. There are tools specifically designed to work with UML diagrams of various kinds, but diagrams can be drawn without them. (In fact — specifically intended to be “whiteboard-friendly”.)

Slide 4

What You (Probably) Already Know About UML

- In PAD II we use UML class diagrams to show things about the classes in a Java program:
  - For each class, its name, attributes (variables), and behavior (methods).
  - Inheritance relationships among classes.
  - Associations between classes — e.g., a ShoppingList class might have an array of ItemAndPrice objects.
- Can become complex, but usually useful in visualizing overall structure of program design.

Slide 5

What UML Can Represent

- Version 2.0 of standard includes many different types of diagrams:

  Class, Sequence, Object, Package, Deployment, Use Case, State Machine, Activity, Communication, Composite Structure, Component, Interaction Overview, Timing

- Quoting from a presentation by Dr. Lewis: "All the diagrams can look pointless or confusing if you don't understand them and use them well." With practice, however, their value starts to become more apparent.
- Most likely to be useful in this course: use case diagrams (for requirements analysis), class and other lower-level diagrams (for design of implementation).

Slide 6

Ways to Use UML

- As a sketch: Throwaway diagrams used to help understand/communicate key features of a design.
- As a blueprint: Detailed diagrams that will be kept as part of system documentation. Probably want to draw these with a tool intended for that purpose.
- As a programming language: Diagrams that can be transformed into code, or inferred from code. (E.g., some IDEs can generate class diagrams from code, and some UML-drawing tools can generate code skeletons from class diagrams.)

Slide 7

UML and Software Development

- Various models for how to build software — Fowler (*UML Distilled*) talks about "waterfall", "iterative", others.
- Four basic phases, though:
  - Requirements analysis.
  - Design.
  - Coding.
  - Testing.

Slide 8

UML Diagrams and Requirements Analysis

- Use case diagrams, and use cases in general.
- Class diagrams (conceptual perspective).
- Activity diagrams (roughly similar to flowcharts).
- State diagrams (roughly similar to state-machine diagrams).

Slide 9

UML Diagrams and Design

- Class diagrams (software perspective).
- Sequence diagrams.
- Package diagrams.
- State diagrams.
- Deployment diagrams.

Slide 10

Why UML in This Course

- For simple programs (such as homework in many courses), it works, sort of, to just focus on code — writing it, or reading it. Often there's not really a requirements analysis phase.
- For larger programs it helps to pay more attention to analysis and design. Doing this in some systematic way helps more, as do pictorial representations. UML is not the only imaginable way to do this, but it's one way.
- Design projects in this course are intended as small-scale versions of "real" development project — so, a chance to practice working in groups, doing a somewhat formal analysis/design, using UML diagrams, . . .

Slide 11

Use Cases, My Two Cents' Worth

- My first reaction — "pointless and confusing". But with repeated exposure, they're starting to look more sensible.
- One articles I found on the Web describes use cases as "stories about how the system is supposed to work."
- Other sources say use cases are as much about writing prose (in a somewhat structured format) as about diagrams. Several of them mention that different formats have been proposed, and there's not one right way, but you should do whatever seems to work.

Slide 12

Requirements Analysis with Use Cases

- Starting point for "real-world" problems — often a vague description. Need to turn this into something specific enough to be a starting point for a design. Many ways, but one that seems to work. . .
- "Use cases" — informally, stories about using a system to meet goals.
- As a running example — ATM. So, we might start one use case like this:

  Withdraw money. A customer arrives at the ATM wanting to withdraw money. The customer inserts his/her card. The system prompts for a PIN . . .

  (We might even just start out with the name of the use case, if it gives enough of an idea.)

Slide 13

Use Cases — Basic Ideas

- Actors — users of the system. Usually humans, but not always.
- Use cases — what happens when actors interact with the system. Together, the use cases should describe all the functionality to be provided.
- Each use case collects possible sequences of actions ("scenarios") relating to a goal.

  ATM example — several possible scenarios for "withdraw money", right? normal withdrawal, incorrect PIN, insufficient funds, etc.

Slide 14

Use Cases — Basic Ideas, Continued

- Probably best to start with prose — for each use case, a name, plus a short description if needed. As analysis continues, fill in details of scenario(s). One format includes:
  - Name (short but descriptive).
  - Main success scenario — sequence of numbered steps, each an element of the interaction between actor and system.
  - Extensions — alternatives for some steps. E.g., if step 2 is "customer enters valid PIN", there would be an extension 2a for "customer enters invalid PIN".

    Or can represent other scenarios as "alternatives".

Slide 15

Use Case — Example Format

Use Case — Withdraw Money

Main Scenario

1. Customer places card in card slot.
2. System prompts for PIN.
3. Customer enters PIN.
4. System verifies that card is valid and prompts user . . . .

. . .

Alternative: Card Rejected

4. System determines card is invalid. . . .

Alternative: Insufficient Funds

Slide 16

Use Cases — HOWTO

- Choose the system boundary (what's inside? what's outside?).

  For ATM, possibly everything that happens inside the box, or inside the box and the network it's connected to.

- Identify primary actors and goals.

  For ATM, actors are customers and service personnel; goals are . . . what?

- Define use cases.
- Draw use case diagram to summarize / present visually.

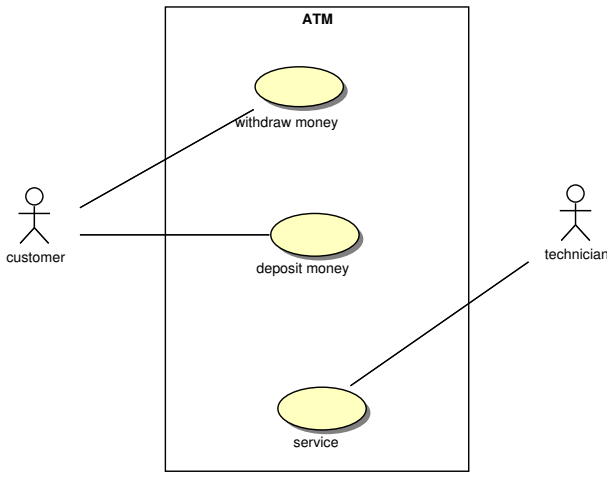
Slide 17

Use Case Diagrams — Basic Ideas

- Draw use cases as ovals, actors as stick figures, system as box enclosing use cases.
- Simplified ATM example . . .

Slide 18

Example



Slide 19

Use Cases — “use” and “include” Relationships

- If multiple use cases share a sequence of identical steps — factor out common steps and use “include”.
- ATM example — **Validate Account** is common to several use cases.

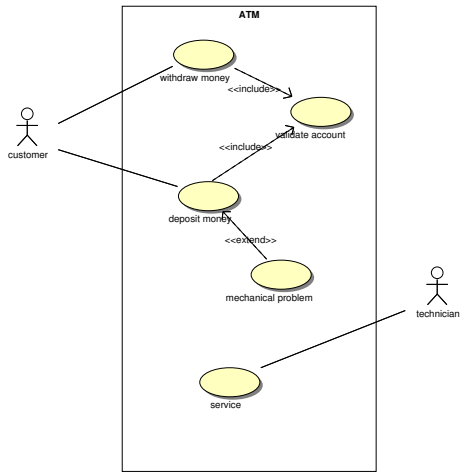
Slide 20

Use Cases — “extend” Relationships

- One way to provide alternatives without modifying existing use case — “extend”. (Note that despite the name this does *not* represent inheritance, but rather a special case that happens sometimes but not always.)
- ATM example — for **Deposit Money**, add **Mechanical Problem** to represent something going wrong with the deposit slot.

Slide 21

Example Revisited



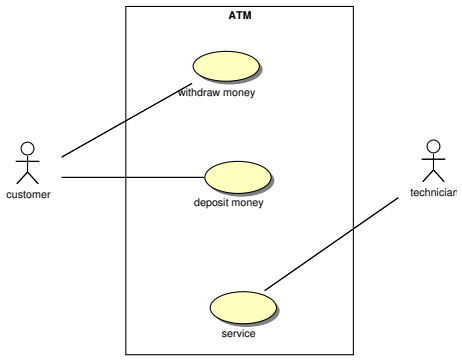
Slide 22

Use Cases — Generalization Relationships

- One way to group alternatives, similar to inheritance in object-oriented language. Specialized versions are aware of general version, not vice versa.
- ATM example — group **Deposit Slot Mechanism Motor Failure** and **Deposit Slot Mechanism Door Failure** as **Mechanical Problem**.

Slide 23

Addition to Example



Slide 24

Use Cases — General Advice

- Try to be somewhat uniform in how you present things, but don't get hung up on details. Goal is to communicate with other humans — fellow designers, customer.
- Can make very detailed diagrams, but probably best not to.