NC STATE UNIVERSITY

Design and Model-View-Controller

CSC216

Supplemental Design Readings

2

NC STATE UNIVERSITY

DESIGN

3

NC STATE UNIVERSITY



Design

- **Goal**: decide the structure of the software and the hardware configurations that support it.
- How individual classes and software components work together in the software system
  - Small programs: ~ 10 classes and interfaces
  - Medium programs: 1000s of classes and interfaces
- Design for single user applications different from web applications and mobile applications
- Software Artifacts: design documents, class diagrams, other UML diagrams

NC STATE UNIVERSITY

OO Design

- Discover classes (and their state)
  - What are the useful entities or concepts in your project requirements?
- Determine the responsibilities of each class
  - What should your class do? How does you class manipulate its data?
- Describe the relationships between the classes
  - Do classes have instances of other classes? Does a method of a class need to use an instance of another class?

NC STATE UNIVERSITY

Finding Candidate Classes and Methods

- Candidate Classes (and state)
  - Nouns
  - You will determine if the noun is an appropriate object or appropriate state
- Candidate Methods
  - Verbs
  - You will determine what class the method is most appropriate for.
- Use CRC cards to help identify candidate classes

6

NC STATE UNIVERSITY

Scholarship Application System

- Faculty and staff can create and edit a scholarship.
- Scholarships have names, an amount, and a list of criteria.
- Students can apply for scholarships.

7

UNIFIED MODELING LANGUAGE

8

UML

- UML: Unified Modeling Language
  - Models object-oriented software
  - Convergence from three earlier modeling languages
    - OMT (James Rumbaugh)
    - OOSE (Ivar Jacobson)
    - Booch (Grady Booch)
- Overseen by the Object Mentor Group (OMG): (www.omg.org)

9

NC STATE UNIVERSITY

Creating UML Diagrams

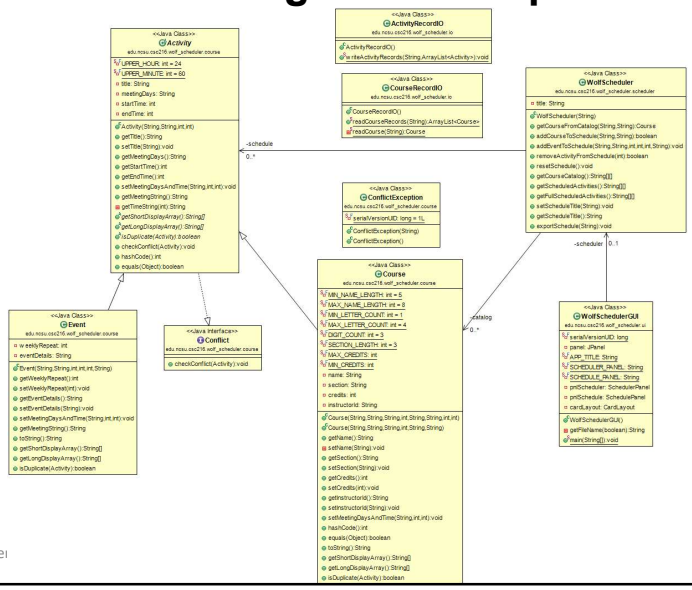
- Commercial or open-source tools
  - Microsoft Visio
  - Commercial Eclipse Plug-ins
  - Dia (freeware)
  - ***UMLetino (web) or UMLet (Eclipse plugin)***
  - PlantUML (“program” diagrams)
  - Draw.io (web)
  - LucidChart (free for students)

NC STATE UNIVERSITY

CLASS DIAGRAMS

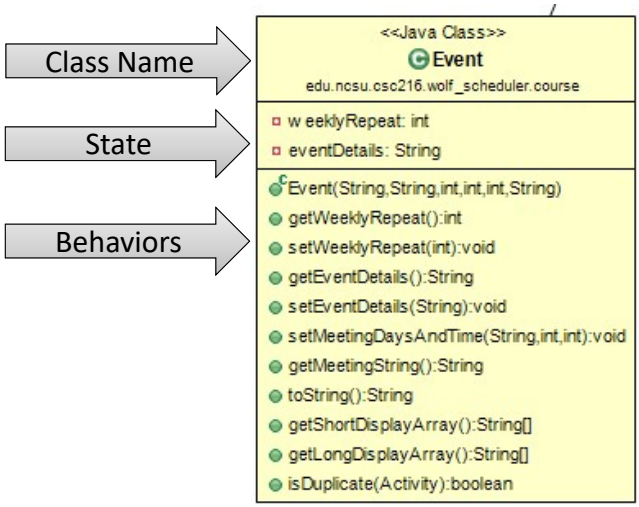
NC STATE UNIVERSITY

Class Diagrams in Depth



NC STATE UNIVERSITY

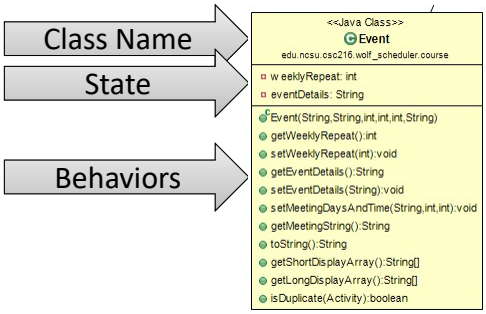
Classes



NC STATE UNIVERSITY

Class Diagram Notation

- Class Names
  - In italics for abstract classes
  - In <> brackets for interfaces (or with a purple I icon)



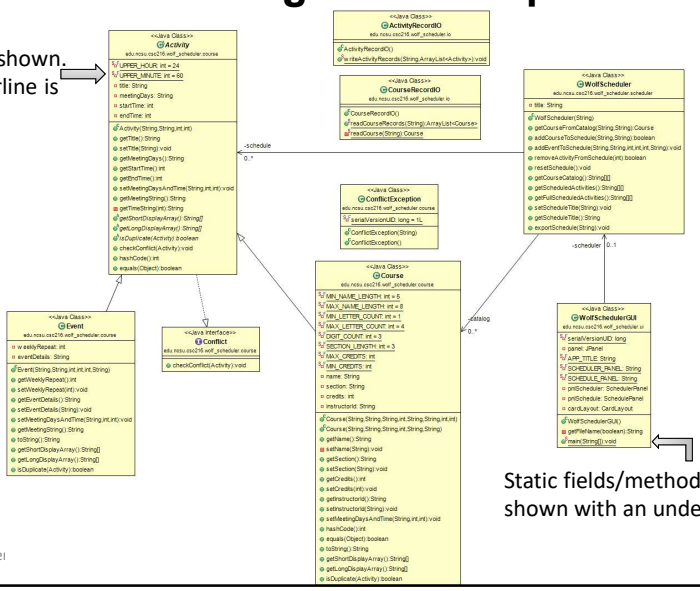
- State & Behaviors
  - Access modifier
    - Private: "-" or red square
    - Public: "+" or green circle
    - Protected: "#" or yellow triangle
    - Default: no symbol
  - Field/method name
  - Behaviors/constructor only: parameter list
  - Return type

```
visibility name[(parameters:type,…)]:type
```


NC STATE UNIVERSITY

Class Diagrams in Depth

Constants have final values shown. B/c they are static, an underline is used.

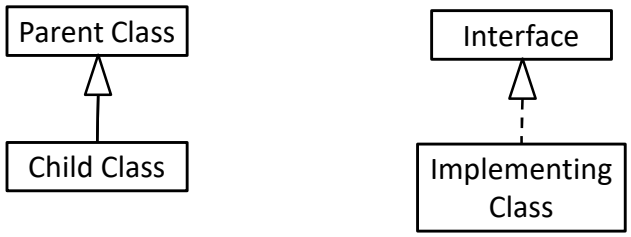


Static fields/methods are also shown with an underline

NC STATE UNIVERSITY

Generalization (is-a)

- Generalization relationships show inheritance
- The child classes inherit the state and behavior of the parent class
  - A `Course` is-a `Event`!
- A dashed line shows a class implementing and interface



16

NC STATE UNIVERSITY

Composition (has-a)

- Composition is when one class has an instance of another class
  - Fields
  - Parameters
- Many ways to model in UML depending on type of composition relationship
- In assignments, requiring a composition relationship means there MUST be a has-a relationship in the diagram
  - It doesn't have to use the composition UML connector of a solid diamond
  - It may be modeled using association (arrow) or aggregation (open diamond) connectors

17

Association Connectors

Indicated by a solid arrow line from the container (source) class to the contained (target) class.



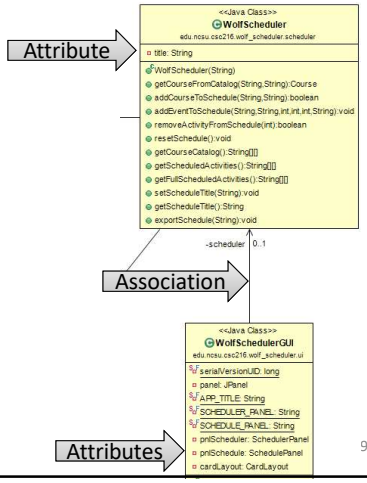
The association object isn't listed as field in container (i.e., Schedule). Instead the name of the field is on the connector (i.e., events).

Can be bi-directional represented by lines without arrow heads



Properties: Attributes vs. Associations

- If attributes and associations are essentially the same thing, when should I use each one?
- Attributes
  - Small things (Strings, primitives, Dates, etc.)
  - Part of existing library
  - Immutable value objects
- Associations
  - Significant classes
  - Part of what will be implemented
  - Mutable references



NC STATE UNIVERSITY

Aggregation vs. Composition Connectors
Whole-Part Relationship

- **Aggregation Connector**: stronger association (unidirectional)
  - Ex. A dept contains a set of employees
  - Ex. A faculty contains a set of teachers
  - A white diamond at the end of the association next to the aggregate class
  - The part can exist separate from the whole

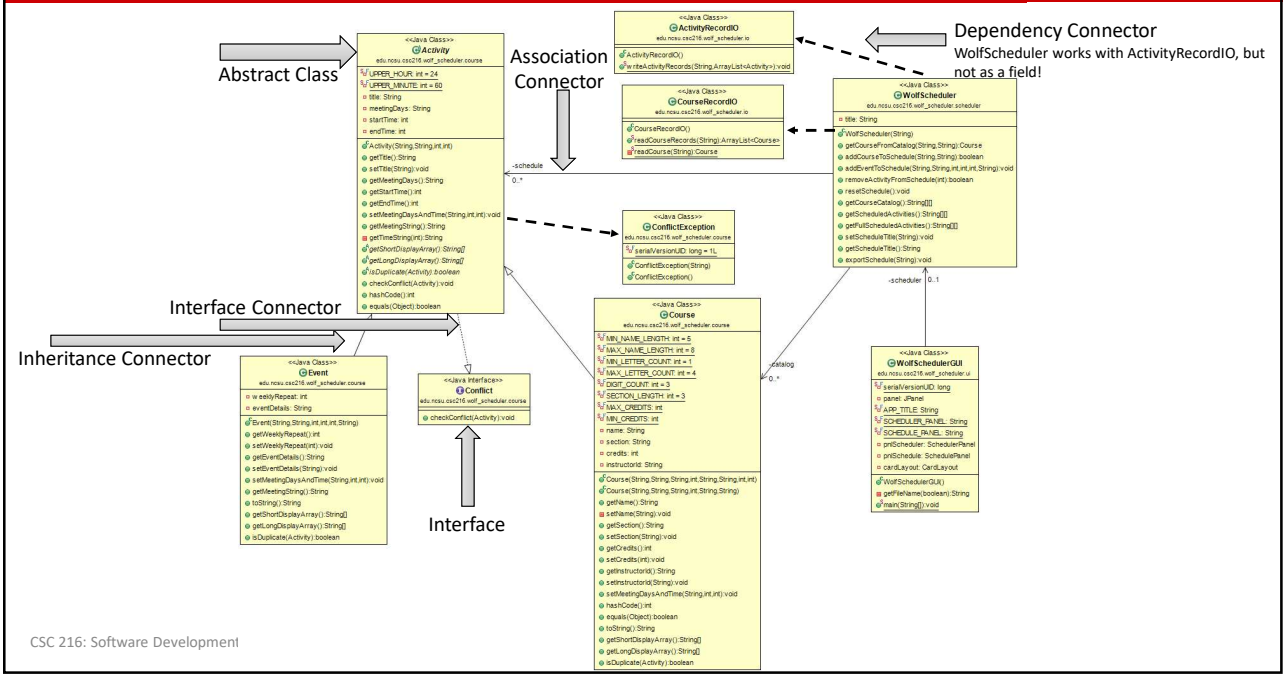


- **Composition Connector**: stronger aggregation (unidirectional)
  - Ex. Invoice– InvoiceLine
  - A black diamond on the end of association next to the composite class
  - The part cannot exist separate from the whole (the whole controls the lifecycle of the part).



20

NC STATE UNIVERSITY

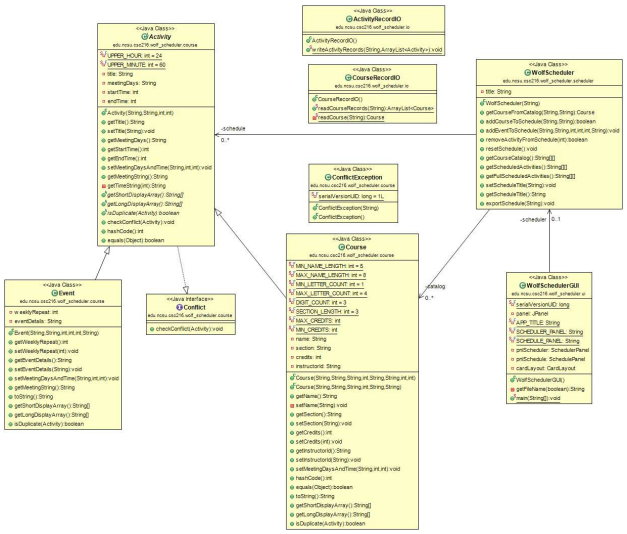


21

NC STATE UNIVERSITY

Interpreting a Class Diagram

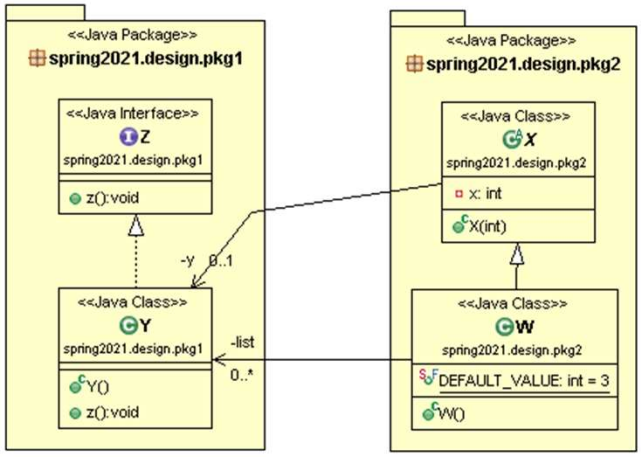
- Activity – abstract class
  - Abstract methods getShortDisplayArray(), get LongDisplayArray(), isDuplicate()
  - Implements Conflict interface
- Conflict – interface
  - Behavior to checkConflict()
- Course – concrete class
  - Extends Activity and (indirectly) implements Conflict
- WolfScheduler – concrete class
  - Has a collection of Activities called schedule
  - Has a collection of Courses called catalog
- WolfSchedulerGUI – concrete class
  - Has an instance of WolfScheduler called scheduler
  - Main method to start program
- ActivityRecordIO – class with static methods
  - Likely used in methods of other classes
- ConflictException – concrete class
  - Likely used in methods of other classes



NC STATE UNIVERSITY

Interpreting a Class Diagram

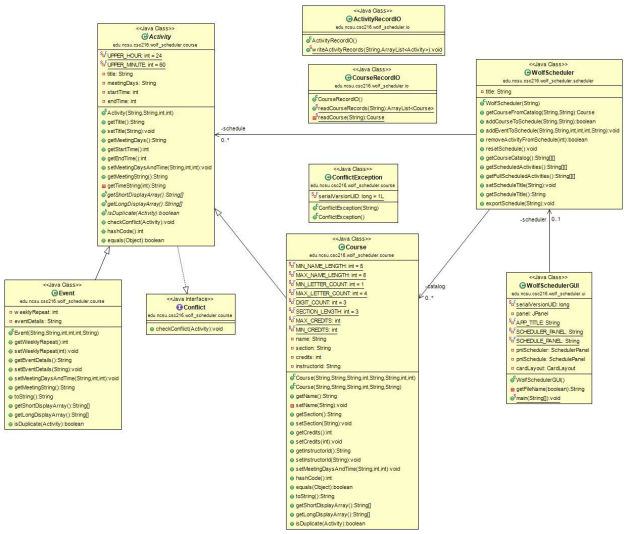
- Describe each of the classes/interfaces in the class diagram



NC STATE UNIVERSITY

Translating a Class Diagram to Code

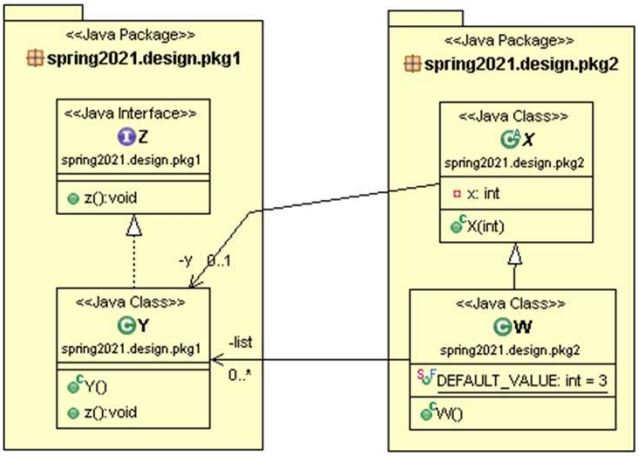
- The icons on the class diagram provide guidelines for creating the skeleton of the program
- Can directly translate the class diagram into code



NC STATE UNIVERSITY

Translating a Class Diagram to Code

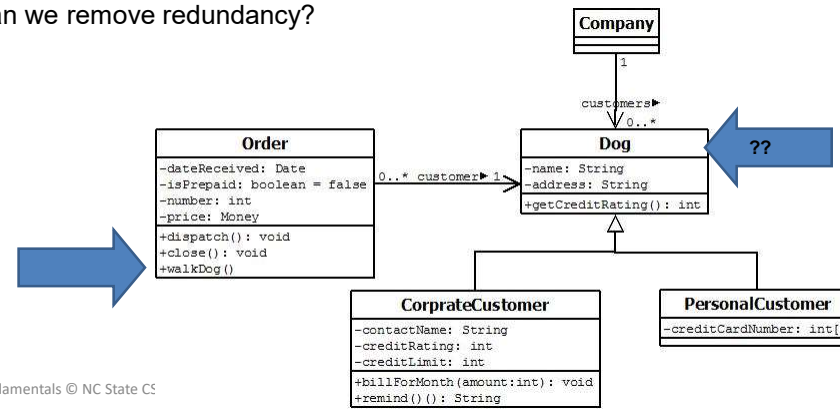
- Using the class diagram, provide a minimal implementation of requested items.



NC STATE UNIVERSITY

Evaluating a Class/Class Diagram

1. Intention-revealing naming: Does the name of the object convey its abstractions? Does the abstraction have a natural meaning and use in the domain?
2. Single Responsibility: Do the name, main responsibility statement data and functions align?
3. Redundancy: Can we remove redundancy?



NC STATE UNIVERSITY

Expectations for Design Proposals and Rationales

- In Part 1s of Projects, you will have the opportunity to propose a design for a set of requirements
- Design Proposal and Rationale – genre of document for computer scientists
  - UML Class Diagram
  - Rationale explaining your design and tradeoffs
- Submitted as a PDF to Gradescope

NC STATE UNIVERSITY

UML Class Diagram

- Show major classes, their state, behaviors, and relationships between them
- Utilizes Model-View-Controller
  - There should be a UI class with no state/behavior other than main (if appropriate for your design)
  - Show how the UI class connects with the model
- Contain at least one interface or abstract class (and the associated relationship that uses it)
  - For interfaces, think about behaviors that one part of the system might expose to another part OR think about one or more behaviors that would be useful outside the system (i.e., Printable is useful for more than just StewRecipes)
- At least one composition relationship. Using the aggregation (arrow) connector is fine!
- Use one of the UML tools suggested earlier (your diagram won't look like the ones in the GPs – those are generated by Eclipse from teaching staff solutions)
- There may be additional expectations depending on the project.

NC STATE UNIVERSITY

Design Rationale

- The Design Rationale explain your UML class diagram and the decisions you made in your design.
  - Objects and their importance – what requirements do they address?
  - Data in the system – how do you know it's needed from the requirements?
  - Responsibilities – are the objects cohesive (i.e., they represent a single abstraction)?
  - Relationships – what are the relationships and why are they important for the design
  - Design patterns – MVC, State Pattern, others – why selected and important
  - Trade-offs – once you make a design decision, you have removed other possibilities and constrained what the system might be able to do given new requirements – describe how the decisions you made impact any future extensions to the system or tradeoffs between two decisions and why you selected the one you did

DESIGN PATTERNS

Design Patterns

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."
  - Christopher Alexander

NC STATE UNIVERSITY

Design Patterns (2)

- Descriptions of communicating objects and classes
  - Participating classes
  - Roles and collaborations
  - Distribution of responsibilities
- Customized to solve a general design problem in a specific context

NC STATE UNIVERSITY

Why Design Patterns?

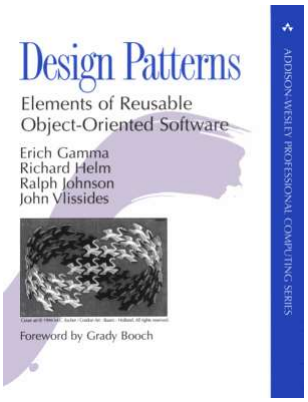
- Don't need to start from scratch to solve a programming problem
  - Reuse previous **useful** ideas
- Design patterns are the closest thing we have to a Design Handbook for software engineering
- Design patterns are not a silver bullet!!! Sometimes the best designs do NOT require design patterns!!!

NC STATE UNIVERSITY

The Gang of Four

- THE book on Design Patterns
- 23 design patterns
- More in literature
- Book focuses on C++, but online references show Java examples

- Patterns are found, not created!



NC STATE UNIVERSITY

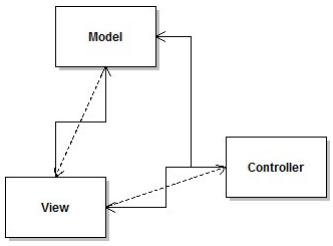
Pattern Families

- **Creational**: process of object creation
  - Singleton
- **Structural**: composition of classes or objects
- **Behavioral**: ways in which classes or objects interact and distribute responsibility
  - State/Strategy
  - Command

MODEL-VIEW-CONTOLLER (MVC)

Model-View-Controller (MVC)

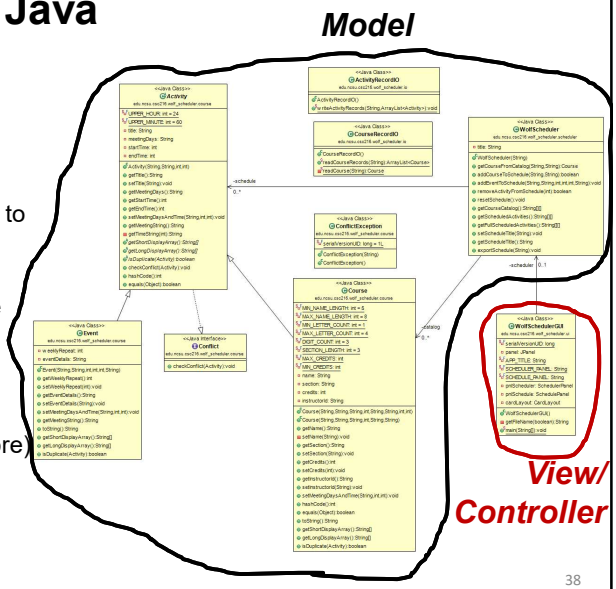
- Isolates business or domain logic from the input and presentation
- **Model**: data underlying the application
  - Changes in state require notifying the view
  - Model abstracts the data storage (DB, POJOs, etc.)
- **View**: UI
  - A model may have many views



- **Controller**: receives input and initiates response by calling model

MVC in Java

- View / Controller: very tightly coupled
  - GUI Applications
    - View (how the object is presented) and controller (how events on the object are processed) belong to one UI object
    - Use generic components (i.e. JButton), which delegate to the underlying “look and feel” (i.e. the operating system)
  - Console Applications
    - View (console prompts) and controller (how user input is processed) typically belong to one (or more) UI objects.
- Model: underlying domain logic



DESIGN BEST PRACTICES

NC STATE UNIVERSITY

Design Best Practices

- Classes should have ***cohesion***
  - The extent to which the code for a class represents a single abstraction
  - Degree to which the members of a class are related to the general purpose of the class
  - Allows for reusability of the class in other programs
- Examples:
  - Student: only contains information relevant to a student
  - Courses: only contains information relevant to courses

NC STATE UNIVERSITY

Design Best Practices (2)

- A program should have low ***coupling***
  - A connection between two classes is a dependency or coupling
    - Instance of an object in the class
    - Call another class to complete a task
  - Internal coupling: modifying another class' data – avoid if possible!
  - Parameter coupling: using services provided by another class – unavoidable
- Highly coupled programs are difficult to write and maintain

NC STATE UNIVERSITY

Design Best Practices (3)

- Data and behavior should be ***encapsulated*** within a class or a package
  - Use packages to group together common functionality
    - Example: In an Android application, all Activities are part of the activity package
  - **Information hiding**: make data members private
  - Details about implementation are hidden within class and only exposed with public members

NC STATE UNIVERSITY

References

- Laurie Williams, UML Class Diagram Slides: http://agile.csc.ncsu.edu/SEMaterials/UMLClassDiagrams.pdf
- http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/
- M. Fowler, UML Distilled, Third Edition, Addison-Wesley, Boston, 2004.