Welcome to CS680!

Tue Thu 7pm - 8:15pm

Course Topics

- Object-oriented design (OOD)
  - Unified Modeling Language (UML)
  - Refactoring
  - Design patterns
  - Object-oriented programming (OOP) with Java
- Continuous testing
  - Automated build of programs
  - Unit testing, static code inspection, etc.
- Basics in functional programming (with Java)
  - Lambda expressions in Java
  - Integration of functional programming (FP) with OOP

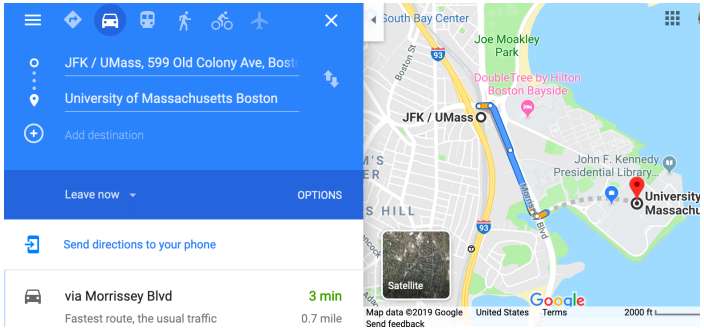
Please Understand…

- You are assumed to be familiar with basic building blocks in OOP.
  - Classes, class instances, data fields, methods, interfaces, inheritance, collections, etc.
- Note that this course DOES NOT cover how to do OOP.
  - You should already know that in sufficient detail.
- Objective in CS680
  - Learn ***design*** and ***organization*** of programs by taking advantage of basic building blocks in OOP.

- Analogy in carpentry:
  - *Objective*: Learn how to design (i.e., draw reasonable blueprints for) things you build (e.g., door case, stairs, deck, kitchen, house).
    - You need to be familiar with basic materials and tools such as:
      - Different types of wood materials
      - Screws, nails, screwdrivers, nail guns, levels, etc.
    - Your focus is to reasonably design a deck, for example,
      - considering proper footings and solid framing to meet given requirements such as size and structural stability.

An Example Scenario

- Your team is expected to develop a navigation app like G Maps.
  - For users to drive and walk (two navigation features)



- How can two sub-groups of the team develop these two features *independently* (i.e. *in parallel*)?
  - How can those 2 features be implemented in a *loosely-coupled* manner?
    - NOT in a tightly-coupled manner.
    - To maximize productivity (development efficiency)
  - How can they be *integrated* at a later project phase in a cost-effective manner?
- How can *something common* be implemented in between the 2 features?
  - Basic data structures, algorithms and UI (e.g. maps, landmarks and shortest-path algorithms)

- Major goal in CS680
  - To be able to answer this kind of questions by learning about ***design*** and ***organization*** of (object-oriented) programs.
- It is easy to say "separating 2 features in a loosely-coupled manner and integrating them later in a cost-effective manner"
- However, it is NOT always that easy to DO it actually.

An Extended Scenario

- Your team is now asked to implement extra navigation features.
  - e.g., with public transportation, a bike, a shared ride, etc.
- How can those extra features be implemented with no/minimum impacts on existing features?
  - How to keep individual features loosely-coupled,
    - so extra features can be introduced in a maintainable and cost-effective manner?

- Goal in CS680
  - To put productivity and maintainability into practice by learning about ***design*** and ***organization*** of (object-oriented) programs.

So, this Course is NOT about

- Learning how to use Java.
- Learning what OOP is and how to do it.
  - What classes, interfaces, inheritance and other basic building blocks in OOP are.
- Learning algorithm design/implementation.

Instead, this course is about

- Learning how to take advantage of those OOP building blocks to develop productive and maintainable (easy to revise/extend) code.

- You will learn that by DOING (i.e. coding).
  - NOT listening.
  - NOT talking.
  - NOT reading.
  - NOT drawing mental pictures.

Textbooks

- No official textbooks.
- Recommended textbooks
  - *Object-Oriented Analysis and Design with Applications (3rd edition)*
    - by Grady Booch et al. (Addison Wesley)
    - General intro to OOAD.
  - *Refactoring: Improving the Design of Existing Code*
    - by Martin Fowler, Addison-Wesley
  - *Head First Design Patterns*
    - by Elizabeth Freeman et al., O'Reilly
  - *Effective Java (3rd Edition)*
    - by Joshua Bloch, Addison-Wesley
  - Fifty Quick Ideas To Improve Your Tests
    - by Gojko Adzic, et al., Neuri Consulting LLP

- The most authoritative and "Bible-like" book on design patterns:
  - *Design Patterns: Elements of Reusable Object-Oriented Software*
    - By Eric Gamma et al., Addison-Wesley

Course Work

- Lectures
- Homework
  - Coding (in Java)
- Grading factors
  - Homework (80%)
  - Quizzes (20%)
    - Occasionally, in lectures.
- No midterm and final exams.

Homework

- There are 2 **FIRM** deadlines for HW submission
  - March 17 (Sun), midnight
    - Submit your solutions for the HWs given by early March.
  - End of the semester
    - Specific date to be announced later.
    - Submit your solutions for the remaining HWs.
  - Firm deadlines are firm. Will NOT grade any late submissions.

- Place your HW solutions at GitHub and email me an invitation to share your repo with me.
  - Share source code (.java files) only, NOT binary code (e.g., .class and .jar files).
  - My GitHub account is jxsboston.
- Learn now how to use GitHub if you are not familiar with it.
- Questions:
  - Email: **Junichi.Suzuki@umb.edu**
  - Office hours: 4pm to 5:30pm, Tue and Thu

Suggestions

- Try your best to turn in your solutions early and regularly as HW assignments are given.
  - Rather than waiting for a deadline to turn them in.
- If you turn in your solutions early…
  - you can ask for my feedback about them, so you can revise them before a deadline.
- If you turn in your solutions regularly…
  - you can earn some extra points.

Departmental Policies on Course Enrollment

- You need a B or better for CS680 to take CS681 and CS682.
- You can't take CS680 more than twice.

Important Notice

- You must work alone from scratch for each HW.
  - You can discuss HW assignments with others. However, DO NOT start with anyone else's code. You must write your code yourself.
- It is an academic crime to
  - Copy (or steal) someone else's code and submit it as your own work.
  - Allow someone else to copy (or steal) your code and submit it as his/her work.
    - Use a private repo to avoid this.
- You will end up with a serious situation if you commit this crime.
  - The University, College and Department have no mercy about it.

- Even if you think you worked on a HW solution yourself, you are placed in a plagiarism case as far as it is somehow identical to someone else's solution.
  - DO NOT start with anyone else's code.

Preliminaries: Unified Modeling Language (UML)

Unified Modeling Language (UML)

- A language to visually ***model*** (or specify) software
  - Intuitively, it is a set of icons, symbols and diagrams to visualize particular elements in software designs.

Customer
- firstName: String - lastname: String
+ getFirstName(): String + getLastName(): String

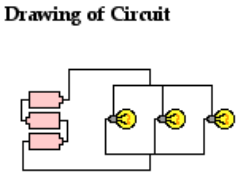
Employee
- name: String - age: int - annualSalary: float
+ setAge(age: int): void

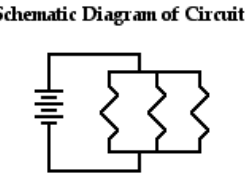
What are Models, and Why?

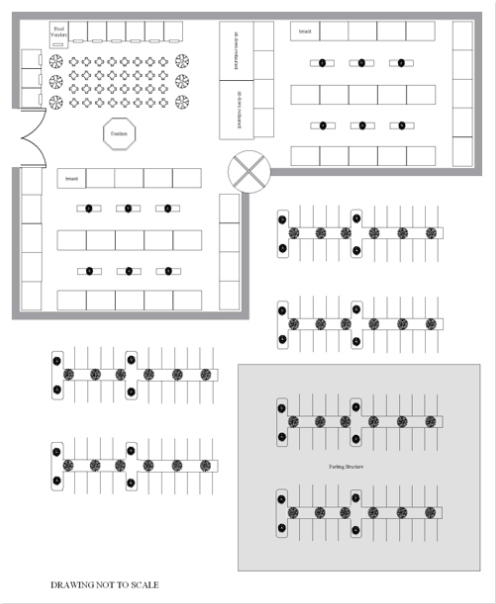
- Visual, intuitive and abstract
  - Makes it easier to describe and understand systems.
  - Good communication tool with the customer and among developers.
- Modeling is a powerful method in any science and engineering disciplines.
  - Biology, chemistry, physics, etc.
    - Modeling atoms, solar systems, etc.

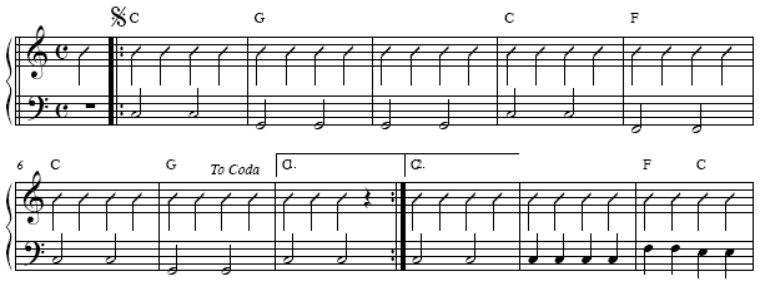
Examples

- 2H + O -> H_2O
- Benzene (molecular formula: C_6H_6)









Boston Red Sox	000 000 300	3
New York Yankees	000 000 000	0

Classes in UML

Customer
- firstName: String - lastname: String - id: int
+ getId(): int

Address
- street: String - city: String - state: String - zipCode: int
+ getStreet(): String + setStreet(street: String): void

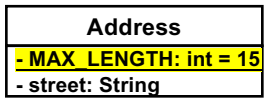
CustomerFactory
+ createCustomer(): Customer

Static methods are underlined.

Just in Case… Static Data Fields and Methods

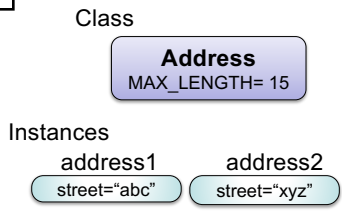
- A static data field
  - Created and used on a per-class basis.
    - All class instances share the data field (and the value in it).
- A regular (non-static) data field
  - Created and used on an instance-by-instance basis.
    - Different class instances have different copies of the data field.

- A static data field
  - Created used on a per-class basis.
    - All instances share the data field (and the value in it).
- A regular (non-static) data field
  - Created/used on an instance-by-instance basis.
    - Different instances have different copies of the data field.



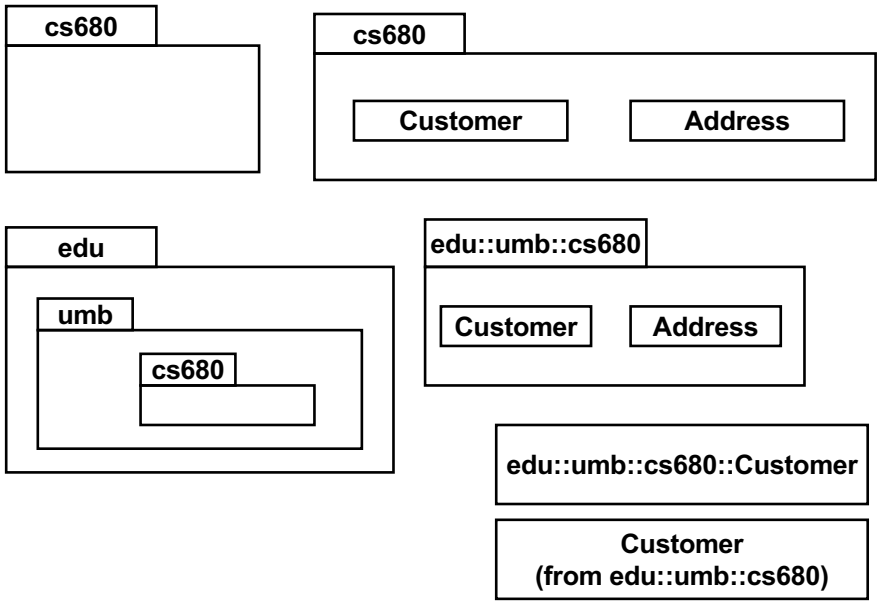

```
public class Address {
  private          String street;
  private static   int MAX_LENGTH = 15;}

Address address1 = new Address(...);
Address address2 = new Address(...);
```

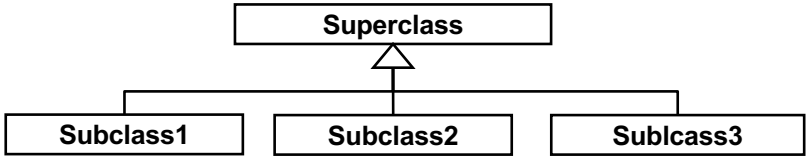


- A static method
  - Created and used on a per-class basis.
  - Can access static data fields.
  - Can NOT access regular (non-static) data fields.
- A regular (non-static) method
  - Created and used on an instance-by-instance basis.
  - Can access both regular (non-static) and static data fields.

Packages in UML



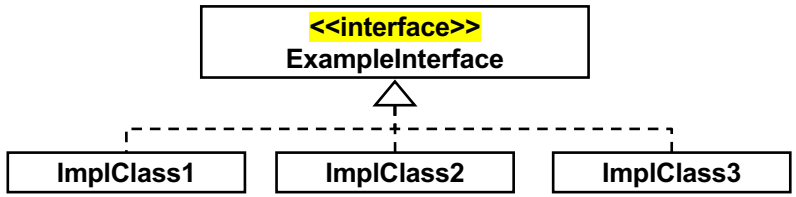
Class Inheritance




```
class Superclass{ ... }

class Subclass1 extends Superclass { ... }
class Subclass2 extends Superclass { ... }
class Subclass3 extends Superclass { ... }
```

Interface-Class Relationship

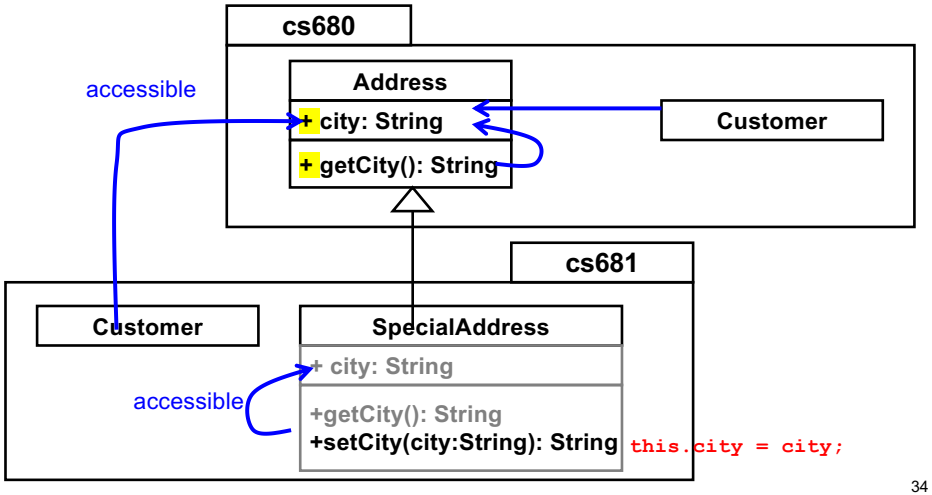


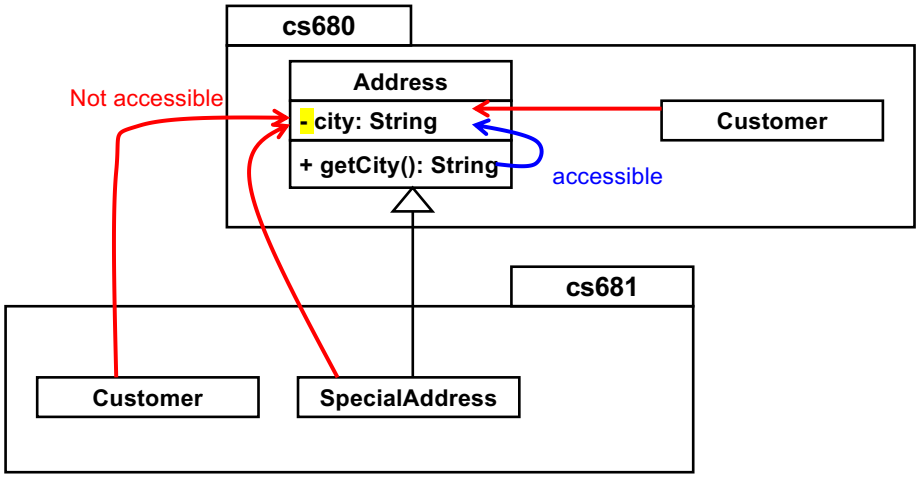

```
interface ExampleInterface{ ... }

class ImplClass1 implements ExampleInterface { ... }
class ImplClass2 implements ExampleInterface { ... }
class ImplClass3 implements ExampleInterface { ... }
```

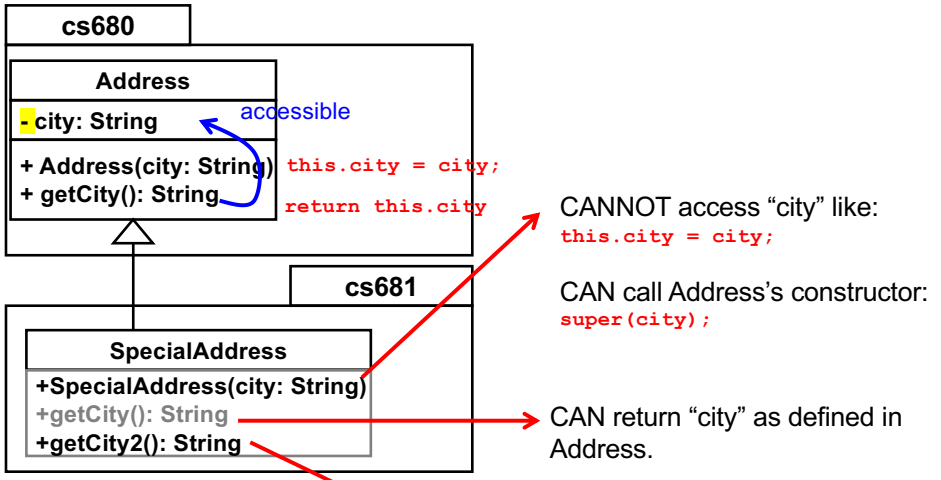

Java's Visibility in UML

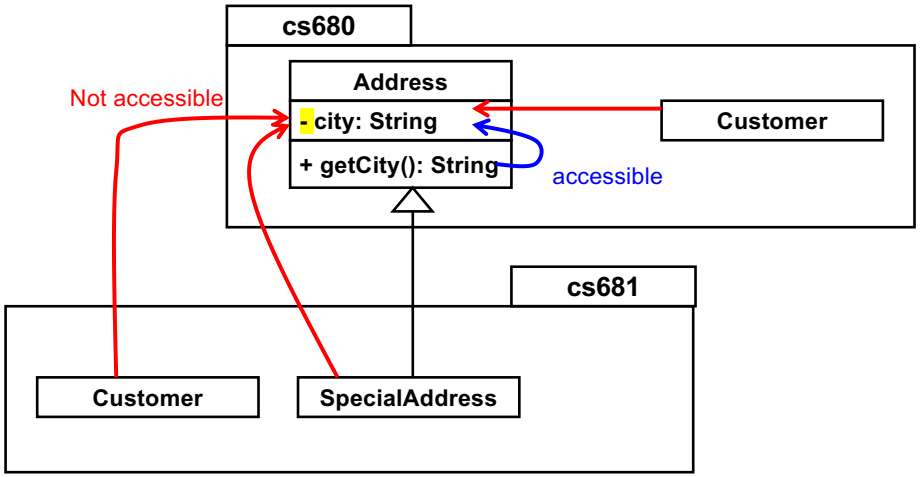
- Indicates who can access a data field or a method
  - Public (+), private (-) or protected (#)



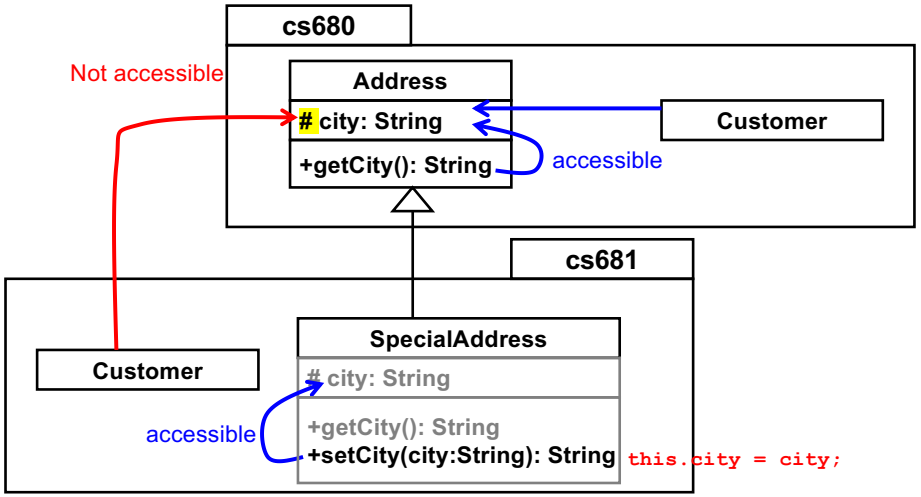


No methods in other classes can access "city" in Address.
No setter methods can be defined in SpecialAddress.

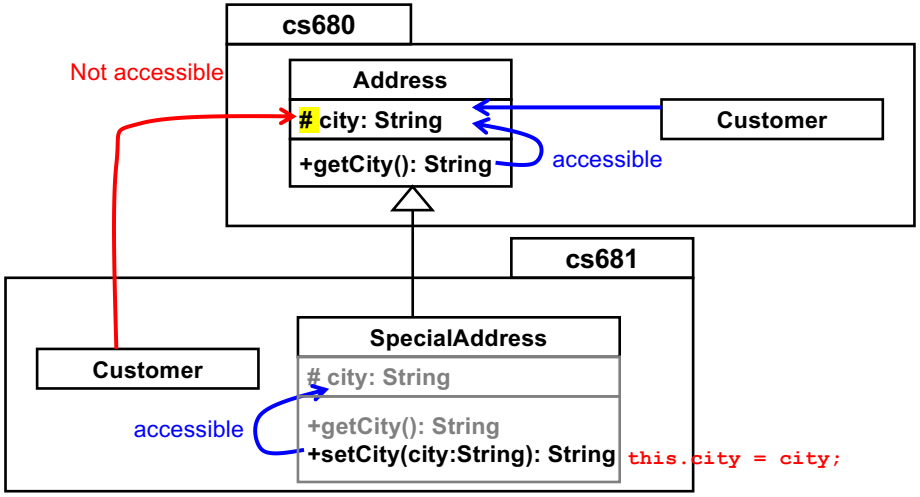




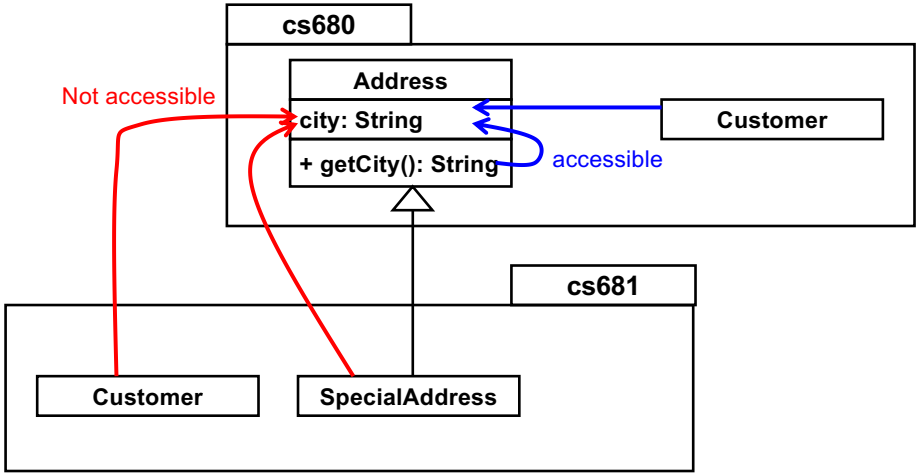
Encapsulation principle: Use private/protected visibility as often as possible to encapsulate/hide the internal data fields and methods of a class.



"City" is visible/accessible in SpecialAddress.
(Both getters and setters can be defined in SpecialAddress.)



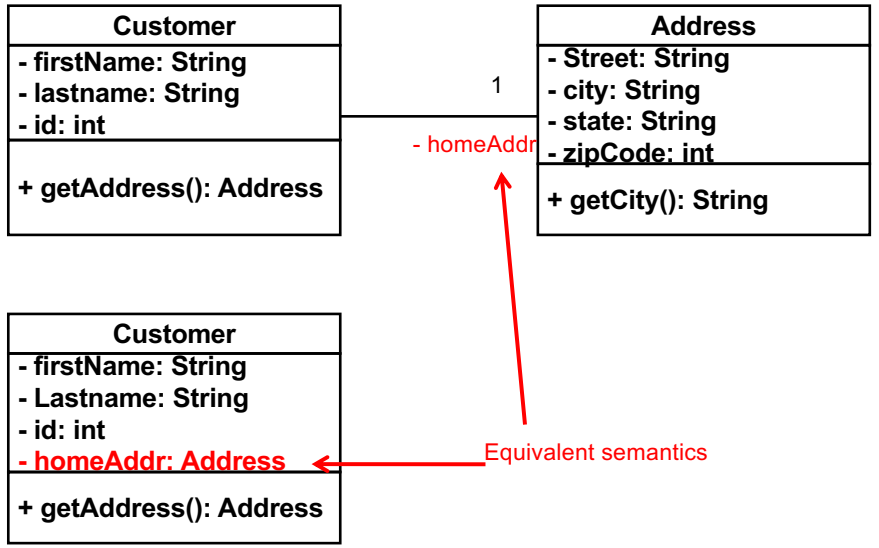
Encapsulation principle: Use private/protected visibility as often as possible to encapsulate/hide the internal data fields and methods of a class.

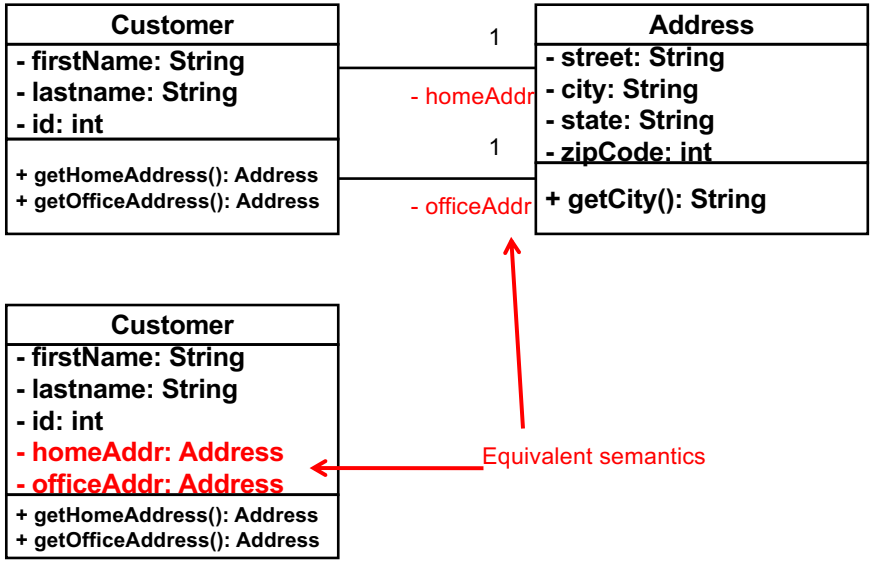


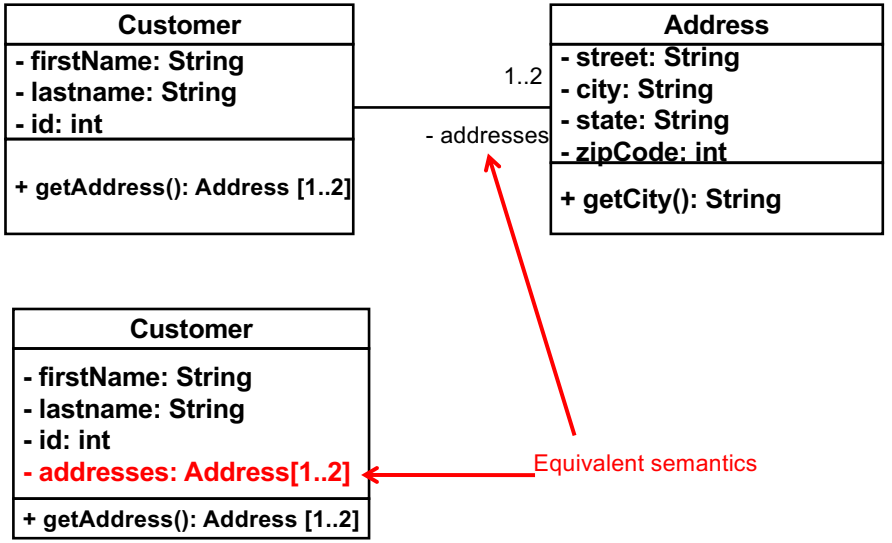
Default visibility (package private) to be used when no modifier is specified.

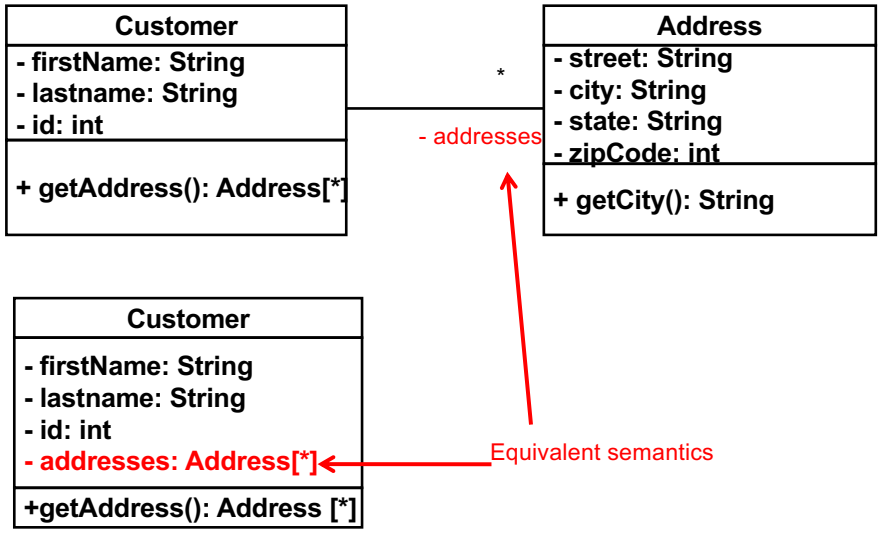
- Specify the visibility for every data field and every method.
- Do not to skip specifying it. (Avoid package-private.)
- It is always important to be aware of the visibility of each data field and method.

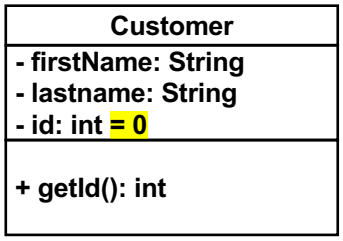
Association











Exercise (not HW)

- Learn about any program elements (e.g. visibility) and UML elements (e.g. association) that you are not that familiar with.
  - Write some code with them and run it.