

Who am I?

- Academics
 - Associate Professor, UMass Boston (2010–)
 - Assistant Professor, UMass Boston (2004–2010)
 - Distributed systems, software engineering and AI
 - www.cs.umb.edu/~jxs/; dssg.cs.umb.edu
 - Post-doctoral Research Fellow, UC Irvine, CA (2000–2004)
 - Ph.D. in Comp Sci from Keio University, Japan (2001)
- Industrial
 - Consultant, cloud computing platform vendor, supply chain mgt. company
 - Tech Director, Object Management Group Japan
 - Co-founder and CTO, TechAtlas Comm Corp, Austin, TX
 - Programmer Analyst, Goldman Sachs Japan
- Professional
 - Member, ISO SC7/WG 19
 - Specification co-lead, OMG Super Dist. Objects SIG

2

Welcome to CS680!

Tue Thu 5:30pm - 6:45pm
Y02-2120

CS680-681-682

- CS680 (F '17):
 - Lecture-based course
 - Object-oriented design
- CS681 (S '18)
 - Lecture-based course
 - Concurrent object-oriented programming
- CS682 (S '18)
 - Project course
- CS 683 no longer exists.
- The department's web pages are not up-to-date.
 - See <https://www.cs.umb.edu/~dsim/cs622/spx.pdf>

Course Topics

- Object-oriented design
 - Design patterns
 - Refactoring
- Continuous delivery/integration (CD/CI)
 - Testing (incl. unit testing, static code inspection)
 - Versioning
 - Automated deployment, configuration mgt., integration testing, etc.
- Lambda expressions in Java
 - Basics in functional programming

Textbooks

- You are assumed to be familiar with object-oriented programming
 - Classes, methods, interface, inheritance, collections, etc.
- Key topics in CS410
 - Design and organization of object-oriented programs
 - An example scenario
 - Your team is expected to develop a navigation app like G Maps.
 - For users to drive and walk (2 navigation features)
 - How can two groups of team members develop the 2 features *independently*?
 - How can those 2 features be implemented in a *loosely-coupled* manner?

5

- 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 Start Design Patterns*
 - by Elizabeth Freeman et al.
 - O'Reilly

6

Course Work

- 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
- Lectures
- Homework
 - Paper reading
 - Modeling/design and coding (Java)
- Individual project

7

8

Grading

- Grading factors
 - Homework (65%)
 - Project deliverables (30%)
 - Quizzes (5%)
 - Occasionally, at the beginning of a lecture
- No midterm and final exams.

My Email Addresses

- Questions → jxs@cs.umb.edu
 - I regularly check this account.
- HW solutions → umasscs680@gmail.com
 - I occasionally check this account.
 - Once a week or so.

9

10

Your Email Address

- Send your (preferred) email address to umasscs680@gmail.com ASAP.
 - I will use that address to email you lecture notes, announcements, etc.
 - You will use that address to submit your HW solutions.

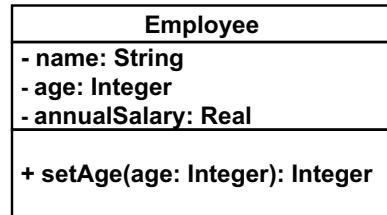
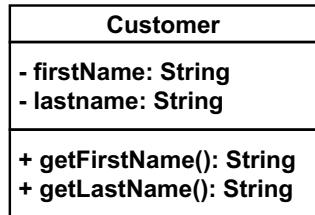
Preliminaries: Unified Modeling Language (UML)

11

12

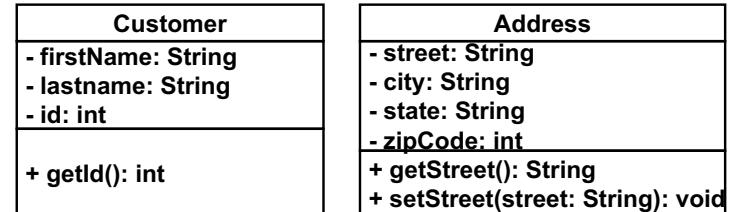
Unified Modeling Language (UML)

- A language to visually *model* software
 - Intuitively, it is a set of icons, symbols and diagrams that denote particular elements in software designs.



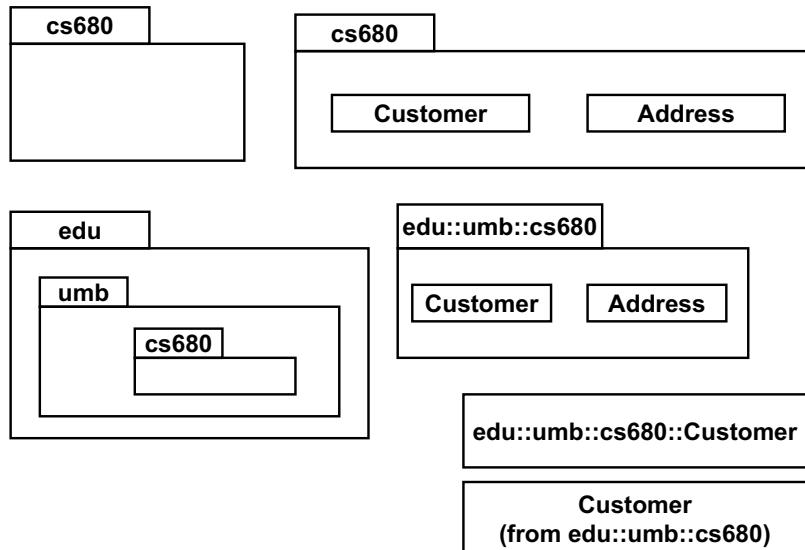
13

Classes in UML



14

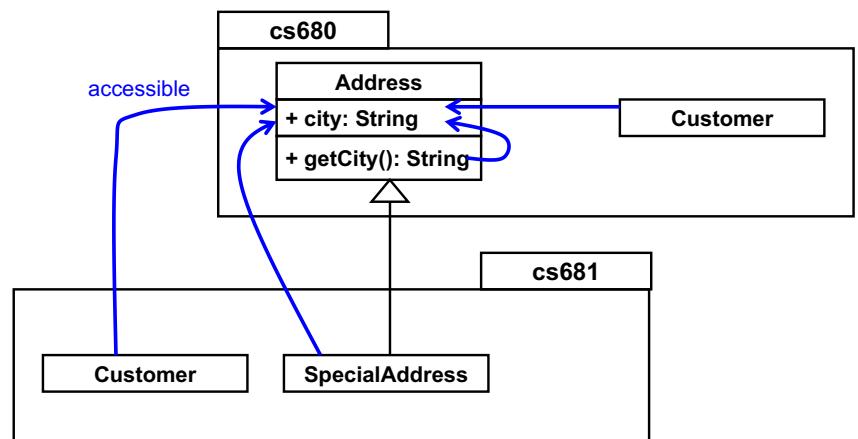
Packages in UML



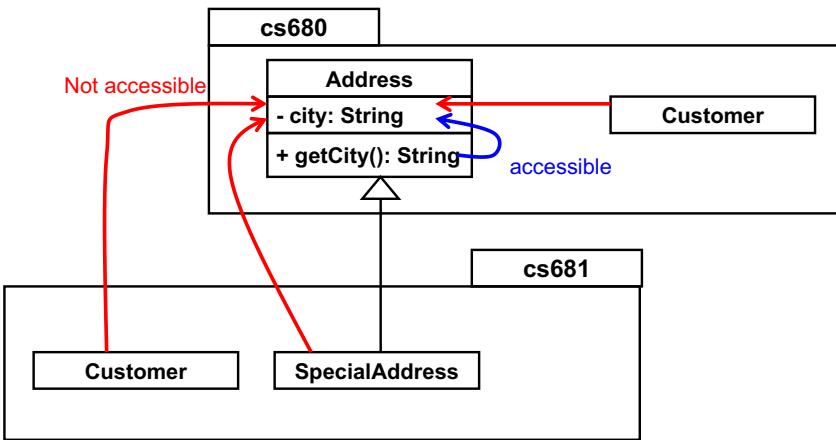
15

Java's Attribute/Op Visibility in UML

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

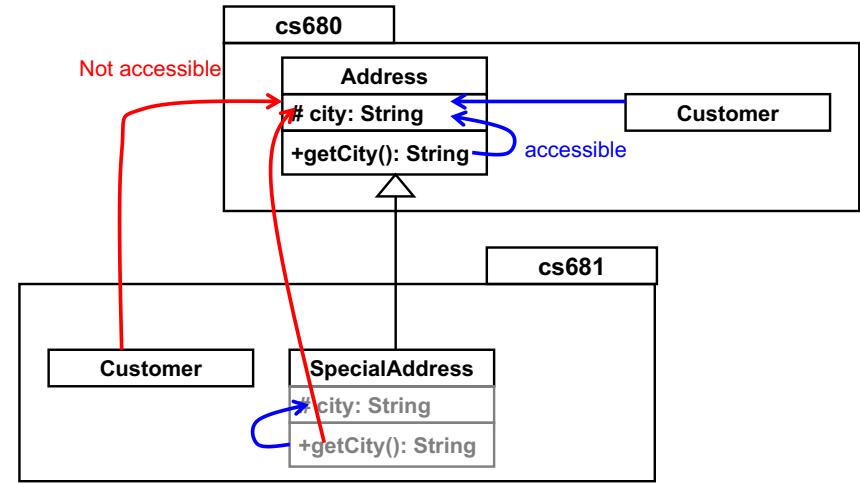


16

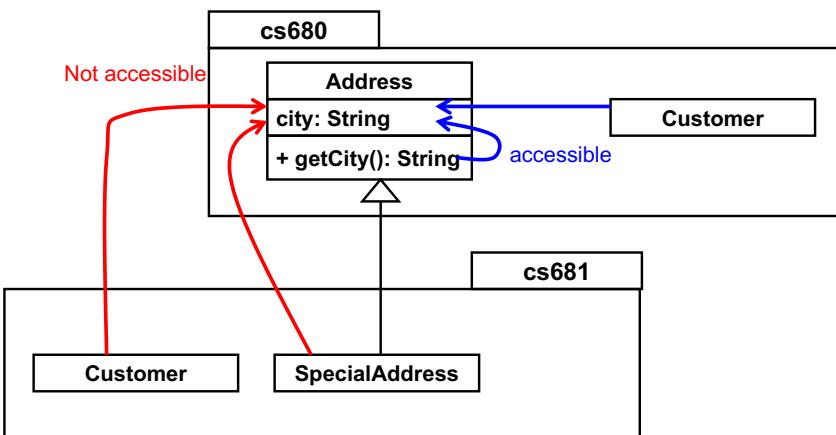


Encapsulation principle: Use private/protected visibility as often as possible to encapsulate/hide the internal attrs/ops of a class.

17



18



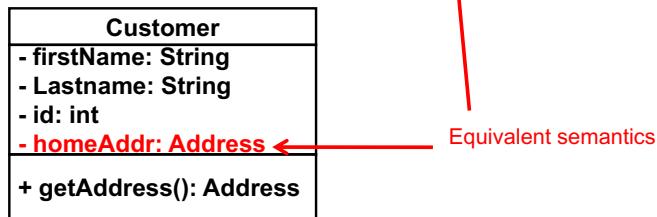
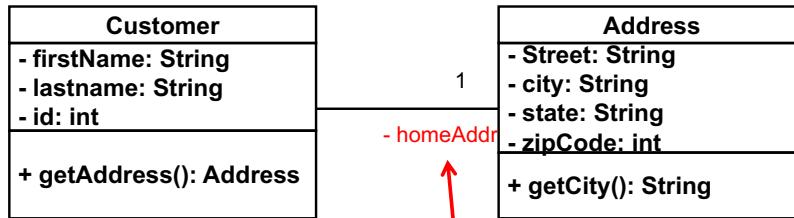
Default visibility (package private) to be used when no modifier is specified.

19

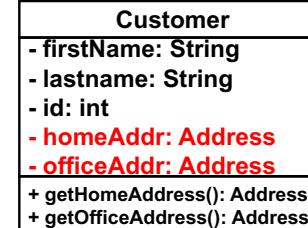
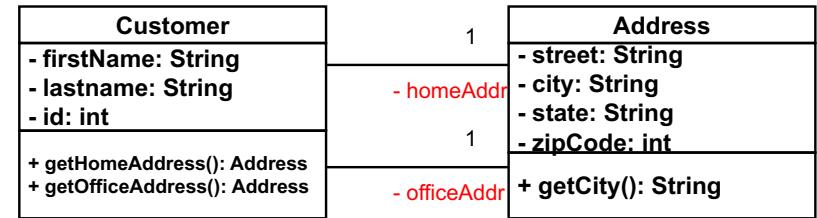
- Specify the modifier for every data field and every method.
- Do not skip specifying it. (Do not use package-private.)
- It is important to be always aware of the visibility of each data field and method.

20

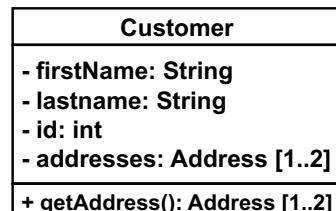
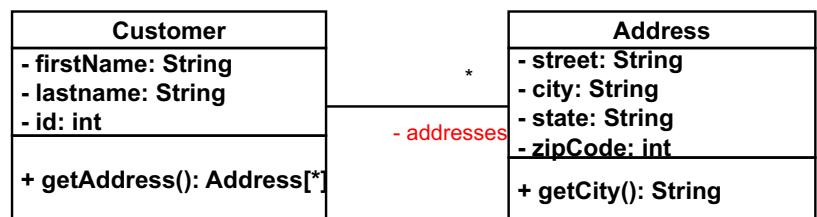
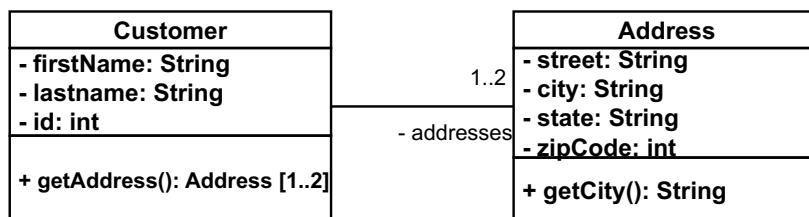
Association



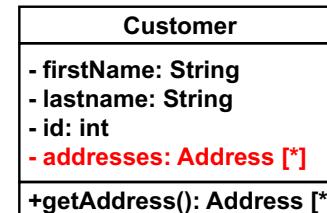
21



22

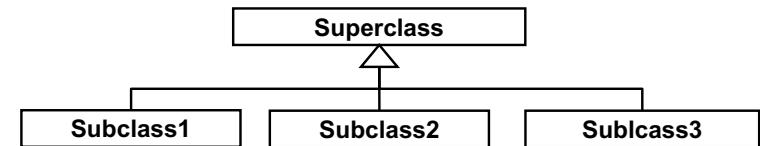
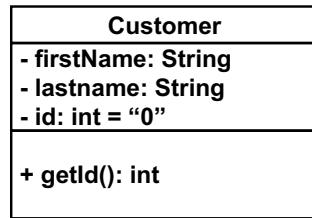


23



24

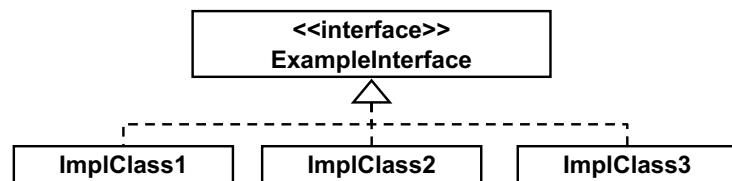
Class Inheritance



25

26

Interface-Class Implementation



Preliminaries:
Road to Object-Oriented Design
(OOD)

27

28

Brief History

- In good, old days... programs had no structures.
 - One dimensional code.
 - From the first line to the last line on a line-by-line basis.
 - “Go to” statements to control program flows.
 - Produced a lot of “spaghetti” code
 - » “Go to” statements considered harmful.
 - No notion of structures (or modularity)
 - Modularity: Making a chunk of code (module) self-contained and independent from the other code
 - Improve reusability and maintainability
 - » Higher reusability → higher productivity, less production costs
 - » Higher maintainability → higher productivity and quality, less maintenance costs

29

Modules in SD and OOD

- Modules in Structured Design (SD)
 - Structure = a set of variables (data fields)
 - Function = a block of code
- Modules in OOD
 - Class = a set of data fields and functions
 - Interface = a set of abstract functions
- Key design questions/challenges:
 - how to define modules
 - how to separate a module from others
 - how to let modules interact with each other

30

SD v.s. OOD

- OOD
 - Intends coarse-grained modularity
 - The size of each code chuck is often bigger.
 - Extensibility in mind in addition to reusability and maintainability
 - How easy (cost effective) to add and revise existing modules (classes and interfaces) to accommodate new/modified requirements.
 - How to make software more flexible/robust against changes in the future.
 - How to gain reusability, maintainability and extensibility?

31

Looking Ahead: AOP, etc.

- OOD does a pretty good job in terms of modularity, but it is not perfect.
- OOD still has some modularity issues
 - Aspect Oriented Programming (AOP)
 - Dependency injection
 - Handles cross-cutting concerns well.
 - e.g. logging, security, DB access, transactional access to a DB
- Highly modular code sometimes look redundant.
 - Functional programming
 - Makes code less redundant.
 - Lambda expressions in Java
 - Intended to make modular code less redundant.

32

Encapsulation

What is Encapsulation?

- Hiding each class's internal details from its clients (other classes)
 - To improve its modularity, robustness and ease of understanding
- Things to do:
 - Always make your data fields private or protected.
 - Make your methods private or protected as often as possible.
 - Avoid public accessor (getter/setter) methods whenever possible.
 - Make your classes final as often as possible.

34

Why Encapsulation?

- Encapsulation makes classes modular (or black box).
 - ```
final public class Person{
 private int ssn;
 Person(int ssn){ this.ssn = ssn; }
 public int getSSN(){ return this.ssn; } }

Person person = new Person(123456789);
int ssn = person.getSSN();
...
```
- What if you find a runtime error about a person's SSN? (e.g., the SSN is wrong or null)... Where is the source of the error, inside or outside Person?
  - You can tell it should be outside Person.
    - A bug(s) should exist before calling Person's constructor or after calling getSSN().
  - You can narrow the scope of your debugging effort.
    - You can be more confident about your debugging.

### Recap

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

35

36

## Violation of Encapsulation

- However, if the Person class looks like this, you cannot be so sure about where to find a bug.

```
- final public class Person{
 private int ssn;
 Person(int ssn){ this.ssn = ssn; }
 public String getSSN(){ return this.ssn; }
 public setSSN(int ssn){ this.ssn = ssn; } }
```

37

- However, if the Person class looks like this, you cannot be so sure about where to find a bug.

```
- final public class Person{
 private int ssn;
 Person(int ssn){ this.ssn = ssn; }
 public String getSSN(){ return this.ssn; }
 public setSSN(int ssn){ this.ssn = ssn; } }
```

```
- Person person = new Person(123456789);
int ssn = person.getSSN();
.....
person.setSSN(987654321);
```

- You or your team mates may write this by accident.
  - It looks like a stupid error, but it is common in a large-scale project.
- Don't define public setter methods whenever possible.

38

## In a Modern Software Dev Project...

- There are a good number of data that don't have to be modified once they are generated.
  - e.g., globally-unique IDs (GUIDs), MAC addresses, customer IDs, product IDs, etc.
- Define them as private/protected data fields.
- No need to define setter methods.

39

- No single engineer can read, understand and remember the entire code base.
- Every engineer faces time pressure.
- Any smart engineers can make unbelievable errors VERY EASILY under a time pressure.
- Your code should be *preventive* for potential errors.

40

# Scale of Modern Software

- All-in-one copier (printer, copier, fax, etc.)
  - 3M+ lines
- Passenger vehicle
  - 7M+ lines ('07)
    - 10 CPUs/car in '96
    - 20 CPUs/car in '99
    - 40 CPUs/car in '02
    - 80+ CPUs/car in '05
      - Engine control, transmission, light, wipers, audio, power window, door mirror, ABS, etc.
      - Drive-by-wire: replacing the traditional mechanical and hydraulic control systems with electronic control systems
      - Car navigation, automated wipers, built-in iPod support, automatic parking, automatic collision avoidance, etc... hybrid cars! autonomous car!!! (e.g. Google's)
- Cell phone (not a smart phone)
  - 10M+ lines

41

- In my experience...
  - 32K, 28K, 25K, 23K, 22K, 20K, 18K, 15K, 12K, 8K, 4K, 3K and 2K lines of Java code for research software
  - 11K and 9K lines of C++ code at an investment bank
  - 7K and 5K lines of C code for research software
- Cannot fully manage (i.e., precisely remember) the entire code base when its size exceeds 10K lines of Java code.
  - What is this class for?
  - Which classes interact with each other to implement that algorithm?
  - Why is this method designed like this?
  - Cannot be fully confident which classes/methods I should modify according to a code revision.
  - Need UML class diagrams for all classes and sequence diagrams for some key methods.
  - Need comments, memos and/or documents about design rationales

42

## Why Encapsulation? (cont'd)

- Assume you are the provider (or API designer) of Person
  - Your team mates will use your class for *their* programming.
  - ```
final public class Person{
    private int ssn;
    Person(int ssn){ this.ssn = ssn; }
    public int getSSN(){ return this.ssn; } }
```
- You can be sure/confident that your class will never mess up SSNs.

43

- However, if you define Person like this,

```
final public class Person{
    public int ssn;
    Person(int ssn){ this.ssn = ssn; }
    public int getSSN(){ return this.ssn; }
    public void setSSN(int ssn){ this.ssn = ssn;} }
```
- You cannot be so sure about potential bugs.

44

- If you define Person like this,

```
- public class Person{
    protected int ssn;
    Person(int ssn){ this.ssn = ssn; }
    public int getSSN(){ return this.ssn; } }
```

- You cannot be so sure about potential bugs.

- However, if you define Person like this,

```
- public class Person{
    protected int ssn;
    Person(int ssn){ this.ssn = ssn; }
    public int getSSN(){ return this.ssn; } }
```

- You cannot be so sure about potential bugs.
- Your team mates can define:

```
- public class MyPerson extends Person{
    MyPerson(int ssn){ super(ssn); }
    public void setSSN(int ssn){ this.ssn = ssn; } }
```

- Your class should be *preventive* for potential misuses.
 - Do not use “protected.” Use “private” instead.
 - Turn the class to be “final.”

45

46

Be Preventive!

- Encapsulation

- looks very trivial.

- is not that important in small-scale (toy) software
 - because you can manage (i.e., read, understand and remember) every aspect of the code base.

- is very important in large-scale (real-world) software
 - because you cannot manage (i.e., read, understand and remember) every aspect of the code base.

Sounds Trivial?

- public class Person{
 private int ssn;
 Person(int ssn){ this.ssn = ssn; }
 public int getSSN(){ return this.ssn; } }

- Once you finish up writing these 4 lines, wouldn’t you define a setter method automatically (i.e. without thinking about it carefully)?

- “I always define both getter and setter methods for a data field. I can delete unnecessary ones anytime later.”
- “Well, let’s define a setter just in case.”
- Think twice. Fight that temptation.
 - Just define the setter method you absolutely need.

47

48

Setters and Getters

- Auto-implemented properties in C# allow you to skip implementing setters/getters explicitly.

```
• class Person{
    public int ssn{get;}
    public String name{get;set;}
    Person(int ssn, String name){
        this.ssn = ssn;
        this.name=name; } }

• Person p = new Person(12345567);
p.ssn;
p.name="Jane Doe";
```

49

- “Access methods” in Ruby allows you to skip implementing setters/getters explicitly.

```
• class Person
    def initialize(ssn, name)
        @ssn = ssn
        @name = name
    end
    attr_reader :ssn
    attr_accessor: name
end

• p = Person.new(12345567, "John Doe")
p.ssn
p.name="Jane Doe"
```

50

- A similar feature is not available in Java. Use lombok (<https://projectlombok.org/>) if you want it.

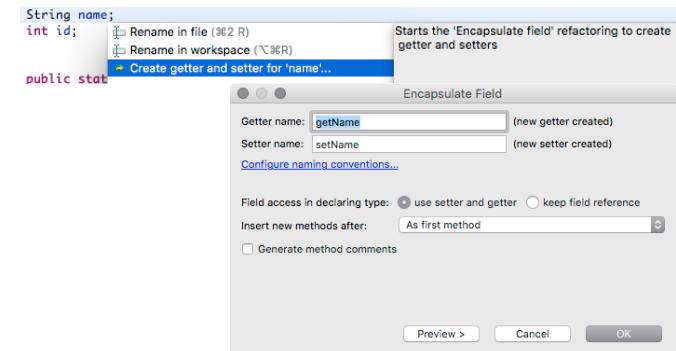
```
• import lombok.AccessLevel;
import lombok.Getter;
import lombok.Setter;
class Person{
    @Getter
    private int ssn;
    @Getter @Setter
    private String name;
    Person(int ssn, String name){
        this.ssn = ssn;
        this.name = name; }
}

• Person p = new Person(12345567, "John Doe");
p.getSSN();
p.setName("Jane Doe");
```

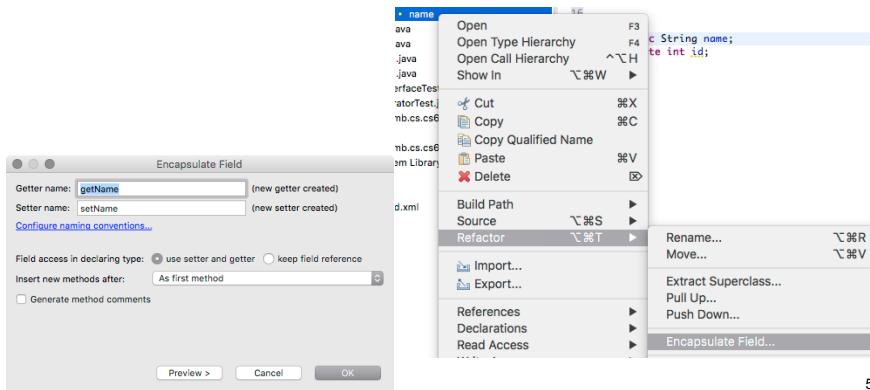
51

Eclipse Tips

- Generate getter and setter methods for a data field.
 - Select a data field’s declaration and perform Quick Assist (Ctrl + 1)

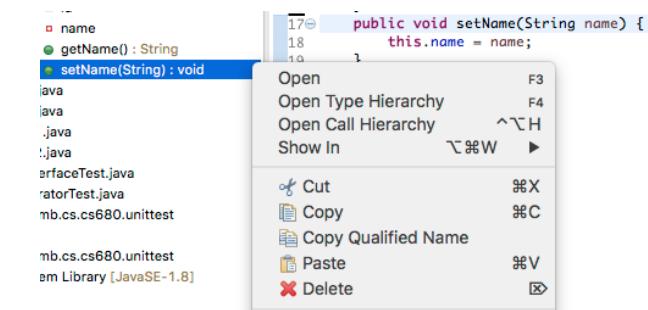


- Encapsulate a public data field.
 - Turn a data field's visibility from public to private
 - Generate getter/setter methods for the data field.



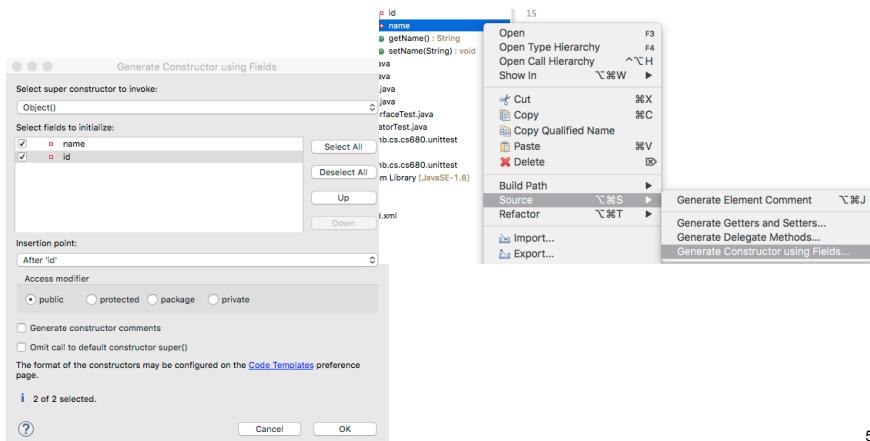
53

- Delete a method



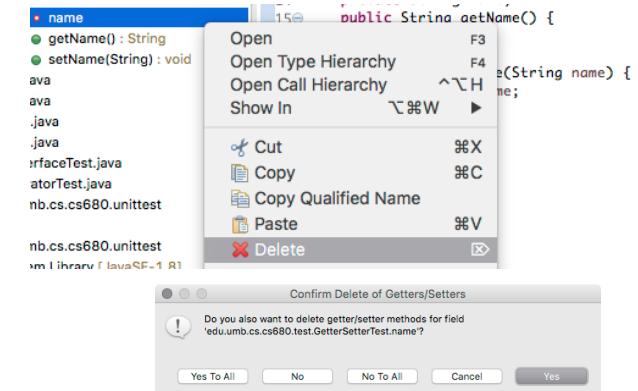
54

- Generate a constructor that accepts a data field(s)



55

- Delete a data field AND its getter/setter methods



56

Exercise

- Write a program based on a given UML diagram
 - Understand a mapping between UML and Java
 - Understand the concept of visibility
 - Understand other keywords in Java (e.g. final)
- An exercise is not a HW. No need to turn in anything for that.