Welcome to CS410!

Tue Thu 2pm - 3:15pm
W-2-200

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, automotive companies
  - 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

Course Work

- Lectures and home work
  - First half of the semester
  - HW: coding in Java
- Group project
  - Second half of the semester
  - Each team works for/with a “customer.”
    - 3 to 5 students a team.
  - Understand what your customer wants.
    - Requirement gathering
  - Deliver a system/product that your customer wants.
  - More details: TBA

Lecture Topics

- Object-oriented design
  - Design patterns
  - Refactoring
- Testing
  - Particularly, unit testing
- Basics in functional programming with Java

- 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?

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

- 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

Grading

- Grading factors
  - Homework (45-50%)
  - Quizzes (0-5%)
    - Occasionally, at the beginning of a lecture
  - Project work (50%)
- No midterm and final exams.

My Email Addresses

- **jxs@cs.umb.edu**

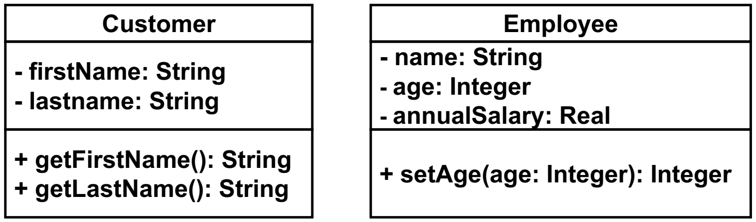
How to Turn in HW Solutions

- Submit source code only.
  - No binaries (No .jar and .class files)
  - More details: TBA
- Where to submit your HW solutions: TBA

Preliminaries: Unified Modeling Language (UML)

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.



Customer

- firstName: String
- lastname: String

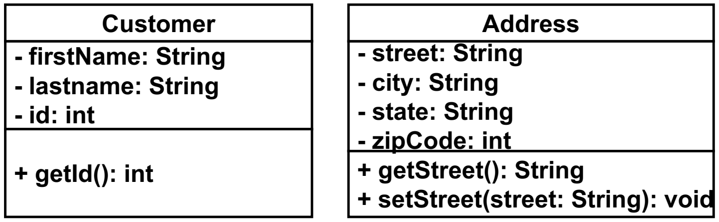
+ getFirstName(): String
+ getLastName(): String

Employee

- name: String
- age: Integer
- annualSalary: Real

+ setAge(age: Integer): Integer

Classes in UML



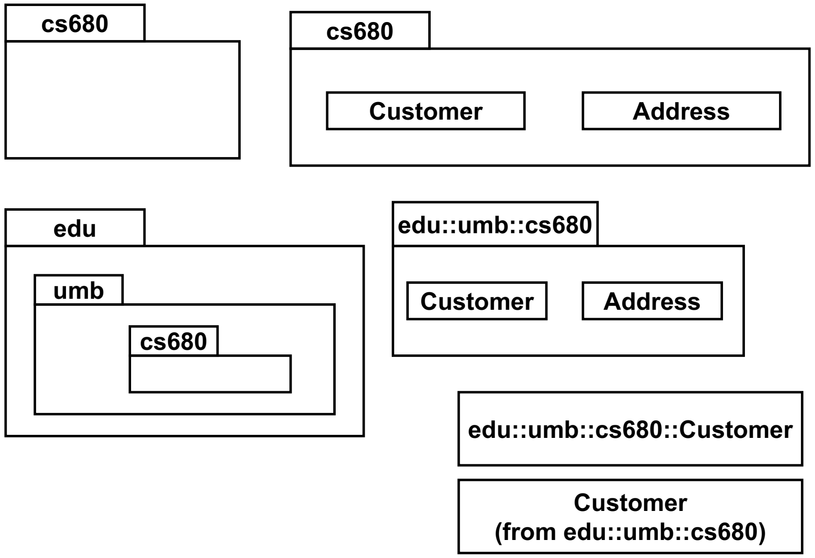
Customer
- firstName: String
- lastname: String
- id: int

+ getId(): int

Address
- street: String
- city: String
- state: String
- zipCode: int

+ getStreet(): String
+ setStreet(street: String): void

Packages in UML



cs680

cs680
Customer
Address

edu
umb
cs680

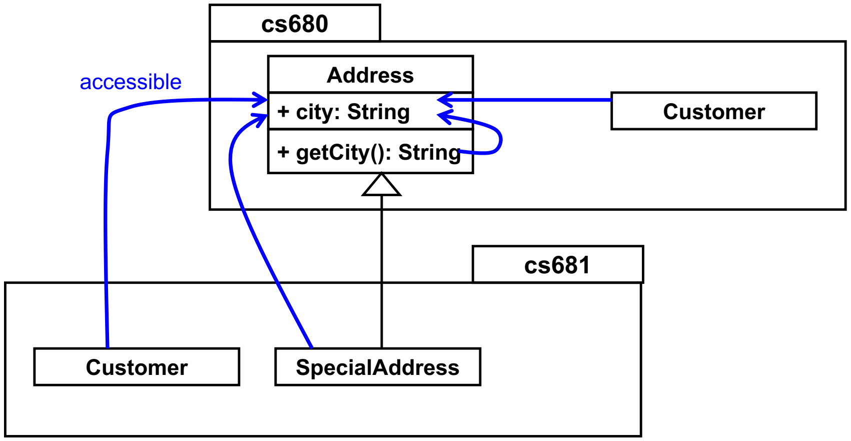
edu::umb::cs680
Customer
Address

edu::umb::cs680::Customer

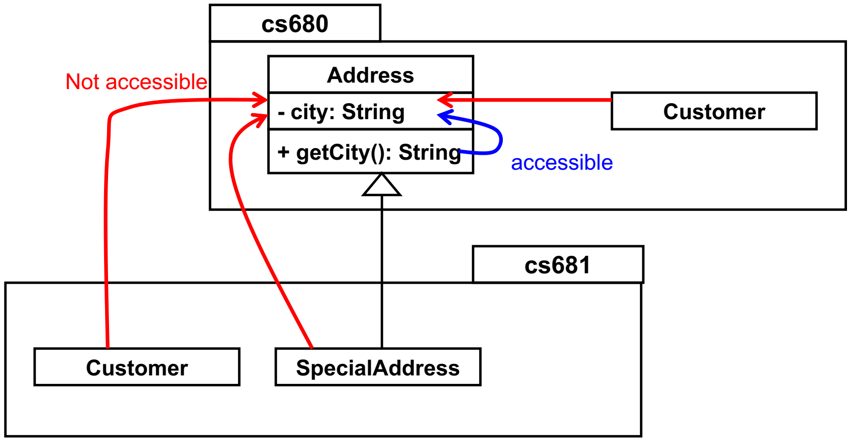
Customer
(from edu::umb::cs680)

Java’s Attribute/Op Visibility in UML

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

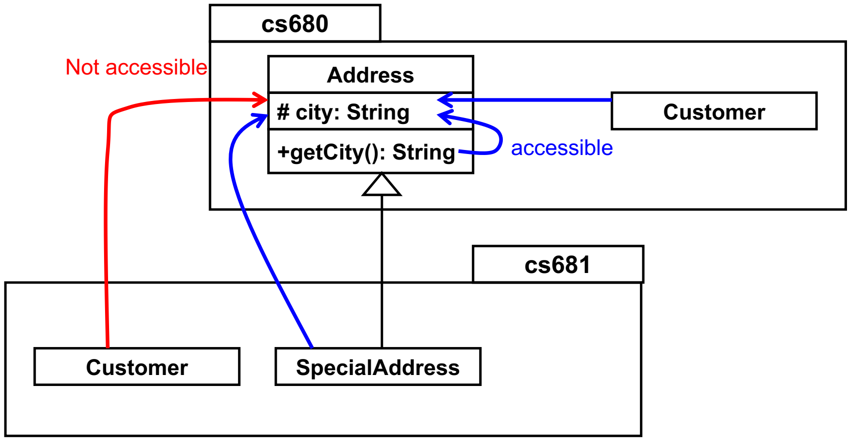


cs680
Address
+ city: String
+ getCity(): String
Customer
accessible
cs681
Customer
SpecialAddress



cs680
Address
- city: String
+ getCity(): String
Customer
Not accessible
accessible
cs681
Customer
SpecialAddress

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



cs680

Address

# city: String

+getCity(): String

Customer

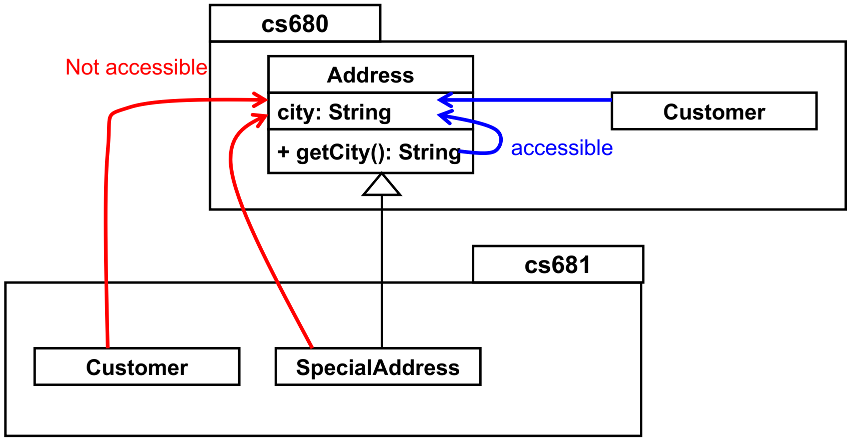
Not accessible

accessible

cs681

Customer

SpecialAddress



cs680

Address

city: String

+ getCity(): String

Customer

Not accessible

accessible

cs681

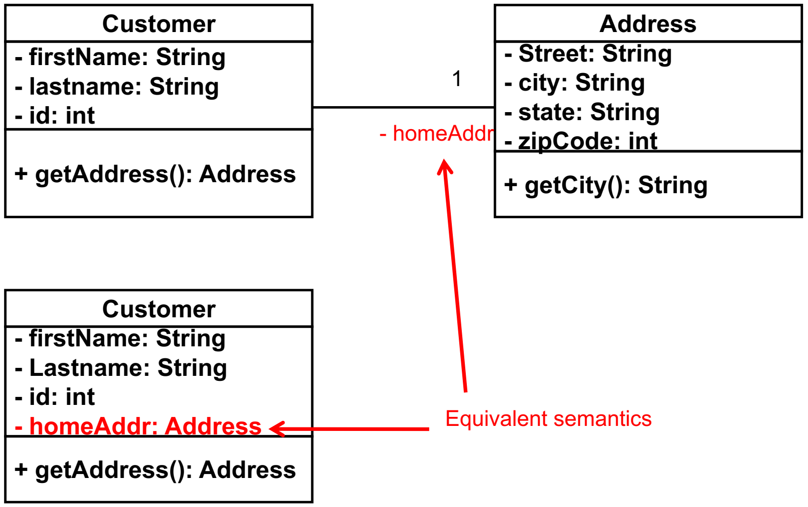
Customer

SpecialAddress

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

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

Association



Customer

- firstName: String
- lastname: String
- id: int

+ getAddress(): Address

1

- homeAddr

Address

- Street: String
- city: String
- state: String
- zipCode: int

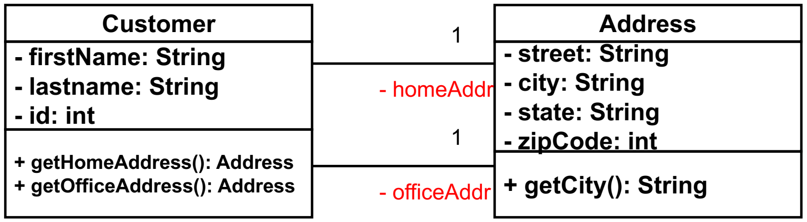
+ getCity(): String

Customer

- firstName: String
- Lastname: String
- id: int
- homeAddr: Address

+ getAddress(): Address

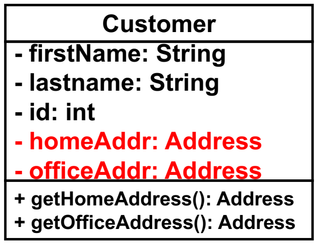
Equivalent semantics



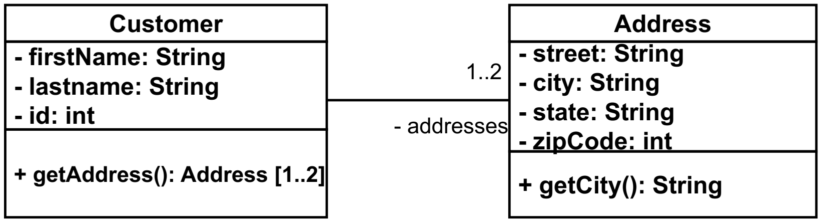
Customer
- firstName: String
- lastname: String
- id: int
+ getHomeAddress(): Address
+ getOfficeAddress(): Address

1
- homeAddr
1
- officeAddr

Address
- street: String
- city: String
- state: String
- zipCode: int
+ getCity(): String



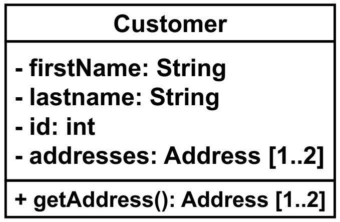
Customer
- firstName: String
- lastname: String
- id: int
- homeAddr: Address
- officeAddr: Address
+ getHomeAddress(): Address
+ getOfficeAddress(): Address



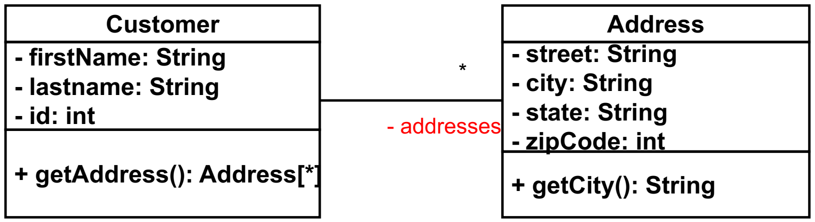
Customer
- firstName: String
- lastname: String
- id: int
+ getAddress(): Address [1..2]

1..2
- addresses

Address
- street: String
- city: String
- state: String
- zipCode: int
+ getCity(): String



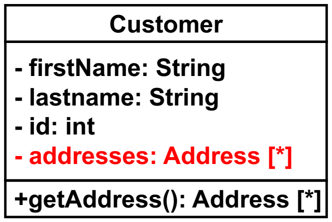
Customer
- firstName: String
- lastname: String
- id: int
- addresses: Address [1..2]
+ getAddress(): Address [1..2]



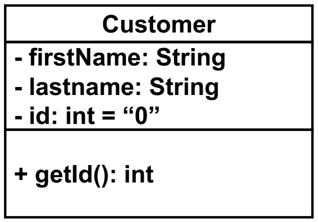
Customer
- firstName: String
- lastname: String
- id: int
+ getAddress(): Address[*]

*
- addresses

Address
- street: String
- city: String
- state: String
- zipCode: int
+ getCity(): String

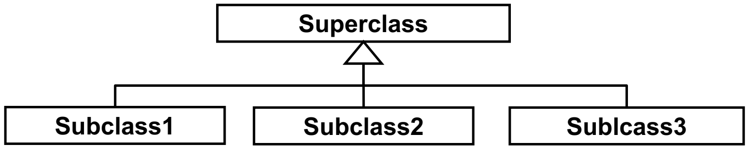


Customer
- firstName: String
- lastname: String
- id: int
- addresses: Address [*]
+getAddress(): Address [*]

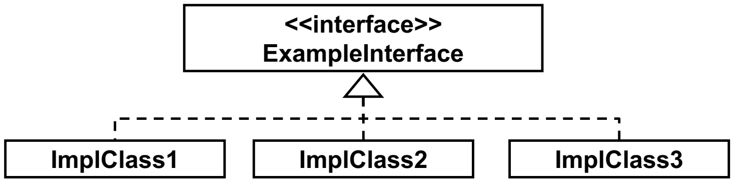


Customer
- firstName: String
- lastname: String
- id: int = “0”
+ getId(): int

Class Inheritance



Interface-Class Implementation



Preliminaries: Road to Object-Oriented Design (OOD)

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

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

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?

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.

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.

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

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; } }
```

- 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.

- 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.

In a Modern Software Dev Project…

- 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.

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

- 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

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.

- 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.

- 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.”

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.

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”;
```

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

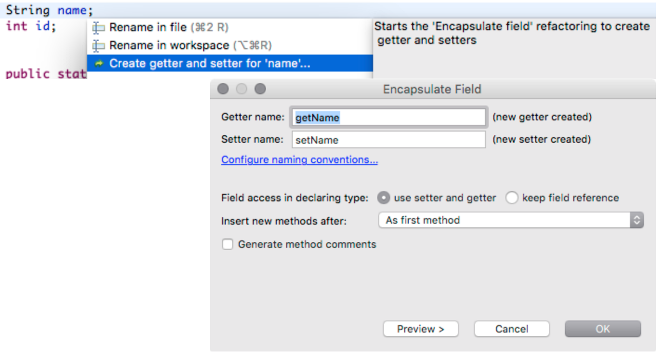
```
• class Person
      def initialize(ssn, name)
          @ssn = ssn
          @name = name
      end
      attr_reader :ssn
      sttr_accessor: name
  end

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

- 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”);
```

Eclipse Tips

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



```
String name;
int id;

public stat
```

Rename in file (⌘2 R)
Rename in workspace (⌥⌘R)
Create getter and setter for 'name'...

Starts the 'Encapsulate field' refactoring to create getter and setters

Encapsulate Field

Getter name: getName (new getter created)

Setter name: setName (new setter created)

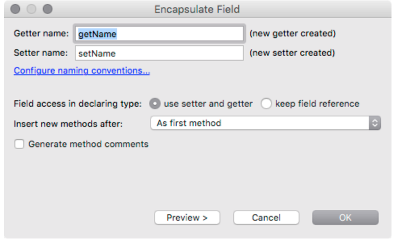
Configure naming conventions...

Field access in declaring type: ● use setter and getter ○ keep field reference

Insert new methods after: As first method

☐ Generate method comments

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



Encapsulate Field

Getter name: getName (new getter created)

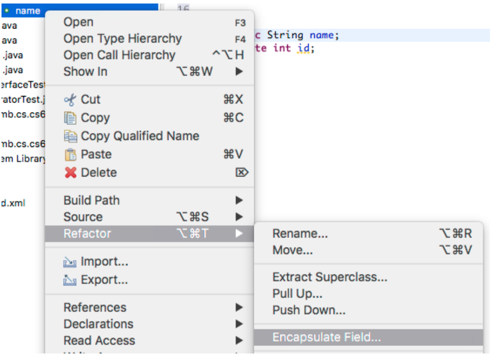
Setter name: setName (new setter created)

Configure naming conventions...

Field access in declaring type: ● use setter and getter ○ keep field reference

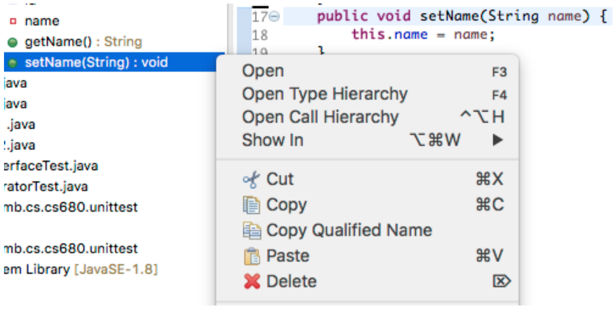
Insert new methods after: As first method

☐ Generate method comments



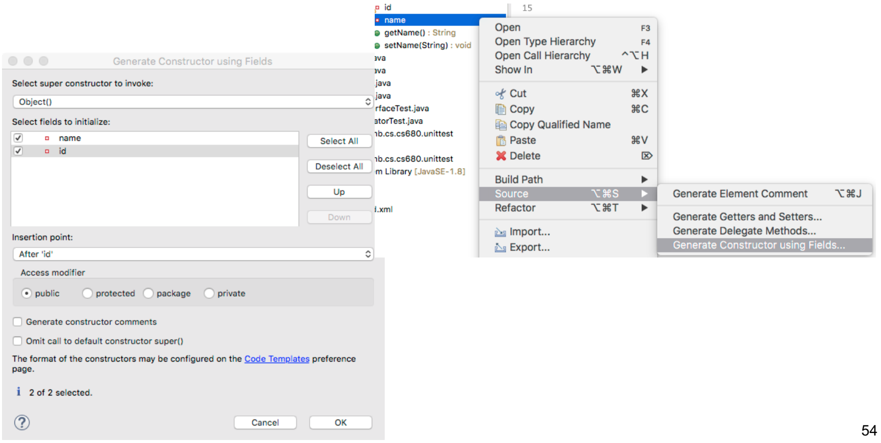
```
c String name;
te int id;
```

- Delete a method



```
public void setName(String name) {
    this.name = name;
}
```

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



Generate Constructor using Fields

Select super constructor to invoke: Object()

Select fields to initialize:

☑ name
☑ id

Insertion point: After 'id'

Access modifier

● public ○ protected ○ package ○ private

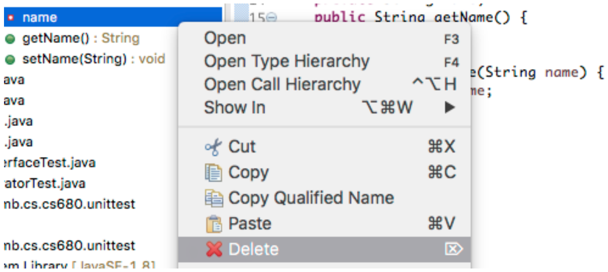
□ Generate constructor comments

□ Omit call to default constructor super()

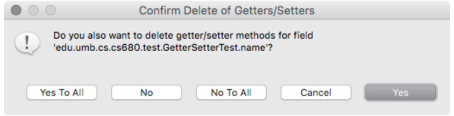
The format of the constructors may be configured on the Code Templates preference page.

2 of 2 selected.

- Delete a data field AND its getter/setter methods



```
public String getName() {
...(String name) {
...me;
```



Confirm Delete of Getters/Setters

Do you also want to delete getter/setter methods for field 'edu.umb.cs.cs680.test.GetterSetterTest.name'?

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.