EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Announcements

→ Course website is up & running on Canvas

→ Course syllabus = Contract for all of us

2

This Week

This Week:
Course Orientation
Delivery Methods, Topics, Assessment, Policies, etc.
"Software is special"

Thursday:
SE History & Ethics

3

About The Instructor

→ Associate Prof, EECS, UC
- PhD, University of Nebraska-Lincoln, 2014
- Assistant/Associate Prof, University of Kentucky, 2014-2020

→ What is my research area?
- Software Engineering
  - Software testing and analysis, mining software repositories, Mobile app development and testing, software architecture, empirical software engineering

→ How to reach me?
- Email: (1) using your UC account; (2) including 'EECE3093C' in the subject
- Office hours: by appointment

4

About This Course

→ Get to know the discipline of software

→ Prerequisite

- Passed CS1022 (CS2) or EECE2040 (data structures programming)
- *Email me if your case is special*

→ Delivery methods: *hybrid*

- First two weeks: Virtual

Tu (asynchronous)	W(virtual)	Th (synchronous)	F(virtual)
Watch video lecture & complete quiz	Lab	**Attend** the *full* online lecture of your *own* section	Lab

- Starting Jan 25: In-person

Tu (in-person)	W(virtual)	Th (in-person)	F(virtual)
Attend the *full* in-person lecture of your *own* section &complete quiz	Lab	**Attend** the *full* in-person lecture of your *own* section	Lab

About This Course

→ Labs will be done online & in groups, with their own tasks, deliverables, and deadlines

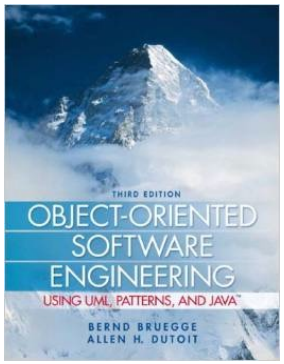
- TAs: Zedong Peng (pengzd@mail.uc.edu), Mona Assarandarban (assarama@mail.uc.edu)

Textbook (as a reference only)

→ Official one

- B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering*, 3rd edition, Prentice Hall, 2010
- Will be used as a reference *only*

→ Supplementary reading/studying materials will be made available through the course website



Source: www.amazon.com

Learning Objectives

→ Object-orientation (OO)

→ Unified Modeling Language (UML)

→ OO design patterns

→ Software testing

→ Collaborative software engineering

→ Software engineering laws

→ …

Assessments

- Quizzes & synchronous-lecture attendance 40%
- Team-based labs 40%
- Final exam 20%

A	[92, 100]	C	[68, 72)
A-	[88, 92)	C-	[64, 68)
B+	[84, 88)	D+	[60, 64)
B	[80, 84)	D	[56, 60)
B-	[76, 80)	D-	[52, 56)
C+	[72, 76)	F	[0, 52)

Important Policies

- Missing or late
  - Require notes/emails/letters ahead of the time
  - Require doctor's notes or other official notes afterward
  - Receive no grade on the assessment (quiz, lab, final exam, etc.) otherwise

What is "software"?

Google software

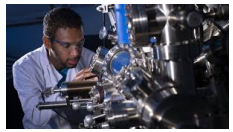
soft·ware
/ˈsôf(t)wer/

noun

the programs and other operating information used by a computer.
"the software industry"

Engineering Software

- Compared to other engineering disciplines, what is *special* about engineering software?
  - Mechanical engineering
  - Electrical engineering
  - Chemical engineering
  - Aerospace engineering




Quiz1 on Canvas

- Completion grade on the "Software is special" question & optional question about "Syllabus"
  - You shall complete these questions & submit the entire Quiz1 ***before*** 11:59pm, Tuesday (Jan 11)
- Instructor's input on the "Week1 attendance" questions
  - This will be done after Thursday's (Jan 13) synchronous lecture, which requires your participation

Summary

- This course is about getting to know the discipline of *software*
  - Delivery method is a combination of online (asynchronous and synchronous) and in-person lectures
    - Know your typical EECE3093C week
  - Course website
    - Get familiar with it & check for updates frequently
  - Assessments
    - Know how your performance will be evaluated
- To-do
  - Complete Quiz1 (Question1) before Tuesday (Jan 11)
  - Make sure to attend Thursday's (Jan 13) synchronous lecture

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

This Week

This Week:
Course Orientation
Delivery Methods, Topics, Assessment, Policies, etc.
"Software is special"

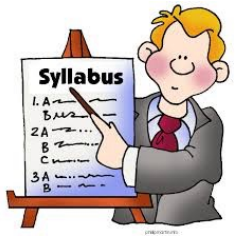
Next Week:
SE History & Ethics

2

Assessments

(in case you missed Quiz1…)

- → Quizzes & in-person/synchronous-lecture attendance 40%
- → Team-based labs 40%
- → Final exam 20%



3

Questions for Syllabus

- → Percentage of quizzes and attendance?
  - ↳ 10% of attendance and 30% of quizzes
- → Attendance of labs
  - ↳ Not mandatory
- → Course schedule
  - ↳ See course calendar
- → Lab teams
  - ↳ Workload is manageable by 2-5 people
- → Final exam
  - ↳ Virtual, 1-hour, open-book/open-internet

4

Delivery Methods

- First two weeks: Virtual

Tu (asynchronous)	W(virtual)	Th (synchronous)	F(virtual)
Watch video lecture & complete quiz	Lab	**Attend** the *full* online lecture of your *own* section	Lab

- Starting Jan 25: In-person

Tu (in-person)	W(virtual)	Th (in-person)	F(virtual)
Attend the *full* in-person lecture of your *own* section &complete quiz	Lab	**Attend** the *full* in-person lecture of your *own* section	Lab

- Or (hybrid)

Tu (asynchronous)	W(virtual)	Th (in-person)	F(virtual)
Watch video lecture & complete quiz	Lab	**Attend** the *full* in-person lecture of your *own* section	Lab

Delivery Methods

- Survey:
  - Would you like to have Tuesdays' classes delivered asynchronously (video lecture, no attendance required) or in-person (attendance required)?
  - This question will be asked in Quiz 2.

What makes software SPECIAL?

- Abstract, invisible, intangible
  - Its purpose is to configure some hardware to do something useful
  - Software is used and experienced
- Never wears out
  - Defects are NOT arising after software has been used often
  - Software follows its own laws other than physical laws
- Is changing all the time
  - Whenever there is a bug, there is a patch
  - Software maintenance costs a lot more than initial builds

Answers from you (N=84)

- Digital vs. Physical
- Flexible (easy to change)
- Brains
- Solve problems in other engineering disciplines
- Need programming language
- Portability
- Continuous delivery

Software-Intensive Systems

→ are engineered to fulfill the *requirements* that are demanded by real *people*

Radiation therapy 

Passenger lift 

Rotterdam barrier 

Car parking 

Flight control 

Cruise control 

9

Software-Intensive Systems

→ are engineered to fulfill the *requirements* that are demanded by real *people*

Industrial press 

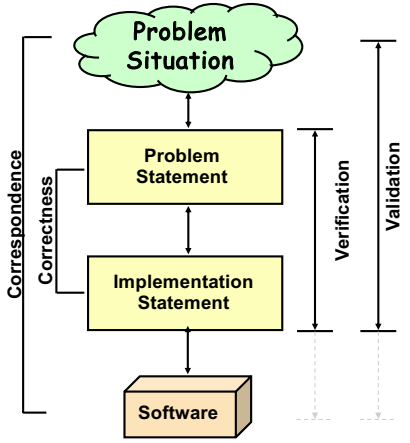
Vending machine 

Medical Records 

Lending Library 

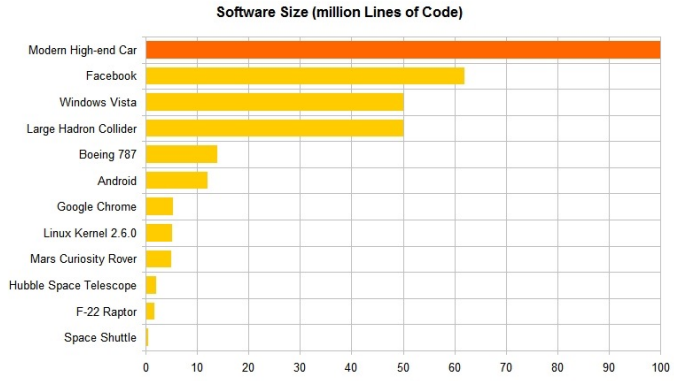
10

Isn't software all about providing a solution?



11

Lines of Code



 Another resource: https://informationisbeautiful.net/visualizations/million-lines-of-code/

12

Summary

- Software is special because
  - it is pervasive
  - it is intangible
  - it is not following physical laws

- To-do
  - Review this week's lecture materials
  - Working on forming teams for the labs

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Course Orientation
"Software is special"

This Week:
SW Eng (SE) History
SE Ethics
Process Models

Next Week:
Intro to OO & Labs

2

Birth of "software engineering"

- Software crisis
  - In the early days of CS
    - ENIAC – first Turing-complete & electronic programmable computer – its construction lasted from 1943-1945
  - Due to the rapid increase in computer power & the complexity of the problems that could be tackled, writing useful and efficient computer programs in the required time became difficult
    - Moore's law – chip performance would double every 18 months – originally stated "a doubling every year" in a 1965 paper
- Software was
  - of poor quality
  - out of budget
  - beyond schedule
  - ...

3

SE was coined in 1968

- To ensure software is built systematically, rigorously, measurably, on time, on budget, and within specification
  - Engineering already addressed such issues, so the same principles used in engineering could be applied to software.
  - "software engineering" was coined in 1968: *rallying cry*
- A flagship event

SOFTWARE ENGINEERING

Report on a conference sponsored by the

NATO SCIENCE COMMITTEE

Garmisch, Germany, 7th to 11th October 1968

4

ICSE http://www.icse-conferences.org

5

What does it take to be a pro?

6

ACM/IEEE Code of Ethics

Full version at https://ethics.acm.org/code-of-ethics/software-engineering-code/ and https://www.computer.org/education/code-of-ethics

- → Aspirations
  - ↳ Without the *details*, the aspirations can become high sounding but empty
- → Details
  - ↳ Without the *aspirations*, the details can become legalistic and tedious
- → Together, the *aspirations* and the *details* form a cohesive code

7

Software Engineer's Aspirations

- → Benefits who?
  1. **PUBLIC:** act consistently with the public interest
  2. **CLIENT AND EMPLOYER:** consistent with public interest
- → What to be disciplined about?
  3. **PRODUCT:** meet the highest professional standards possible
  4. **JUDGMENT:** maintain integrity and independence
  5. **MANAGEMENT:** lead by examples & in an ethical way
- → Be responsible for whom?
  6. **PROFESSION:** advance the profession's reputation
  7. **COLLEAGUES:** be fair and supportive
  8. **SELF:** lifelong learning regarding the practice

8

Details of "act in line with public interest"

1.01 Accept full responsibility for their own work.

1.02 Moderate the interests of the software engineer, the employer, the client and the users with the public good.

1.03 Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.

1.04 Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.

9

Details of "act in line with public interest"

1.05 Cooperate in efforts to address matters of grave public concern caused by software, its installation, maintenance, support or documentation.

1.06 Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools.

1.07 Consider issues of physical disabilities, allocation of resources, economic disadvantage and other factors that can diminish access to the benefits of software.

1.08 Be encouraged to volunteer professional skills to good causes and contribute to public education concerning the discipline.

10

Selected Details

Judgment (6 details)	Management (12 details)
Only endorse documents either prepared under their supervision or within their areas of competence and with which they are in agreement.	Ensure that software engineers are informed of standards before being held to them.
Maintain professional objectivity with respect to any software or related documents they are asked to evaluate.	Assign work only after taking into account appropriate contributions of education and experience tempered with a desire to further that education and experience.
...	...

11

Case Study: To ship or not to ship?

Rachel works as a Quality Assurance Engineer at a large electronics company. She is responsible for the final testing of her company's servers and is part of a team that decides when new products will be shipped to distributors for sale. Lately, the business model requires the release of a new generation of servers approximately every six months, meaning that Rachel has a limited timeframe to conduct her Quality Control tests (that is, performing every possible test on the servers is impossible). Rachel will not ship a product if there is any possibility that the server could malfunction and cause physical harm to the customer. However, she will ship a product that has a higher likelihood of failure resulting in data loss for the customer, because she knows that if she doesn't, her company's competitor will. Is this an ethical way to conduct business?

 Source: https://www.scu.edu/ethics/focus-areas/more/engineering-ethics/engineering-ethics-cases/computer-engineering-cases/

12

Quiz Question #1: Ethics

→ Explicitly relate to *at least two* SW Eng. Code of Ethics *aspirations*

- https://www.computer.org/education/code-of-ethics
- https://ethics.acm.org/code-of-ethics/software-engineering-code/

to argue why, or why not, Rachel's way to conduct the business is ethical.

- You are encouraged to use *details* within each aspiration to support your argument; however, you must cover at least two *aspirations*.
- Please *begin your answer* by listing explicitly the aspirations (and details, if applicable) that you use.

Quiz Question #2: Group Formation

- Each group shall have 2 to 5 people
- Yes/No question: "I need help from the instructor to form a team"
  - If you answer "No", you must ask one member to email the instructor (tingting.yu@uc.edu) before 11:59pm, Friday (Jan 21) by CC-ing all the group members with their UC email account (i.e., the one with the domain name: mail.uc.edu). Please use "EECE3093C Team Formation Information" as the email subject, and list in the email body the full name and the section number of each group member.

Quiz Question #3: Delivery Methods

→ Survey:

- Would you like to have Tuesdays' classes delivered asynchronously (video lecture, no attendance required) or in-person (attendance required)?

Quiz Question #4: 1/20 attendance

→ You shall leave this question *unanswered*, and submit the entire quiz before 11:59pm, Tuesday (1/18)

Summary

→ SE was coined in 1968

- Combat "software crisis"

→ SE Code of Ethics (in the "Basics" folder)

Full version at https://ethics.acm.org/code-of-ethics/software-engineering-code/
and https://www.computer.org/education/code-of-ethics

- Aspirations & details
  - Benefits who? Disciplined about what? Responsible for whom?

→ To-do

- Complete Quiz2 before Tuesday (Jan 18)
- Make sure to attend Thursday's (Jan 20) synchronous lecture
- Email your team information to me before Friday (Jan 21)

EECE3093C: Software Engineering

(Spring 2022)

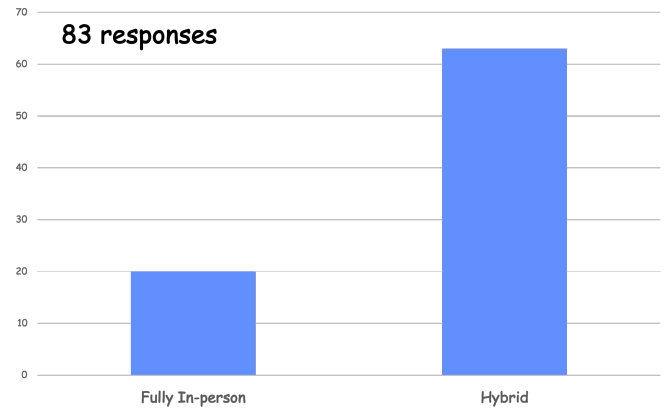
Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Course Delivery Method



2

Course Delivery Method: Hyrbid

Tu (asynchronous)	W	Th (in-person)	F
Watch video lecture & complete quiz	Lab	**Attend** the *full* in-person lecture of your *own* section	Lab

3

Today's Menu

Last Week:
Course Orientation
"Software is special"

This Week:
SW Eng (SE) History
SE Ethics
Process Models

Next Week:
Intro to OO & Labs

4

From Tuesday to Today

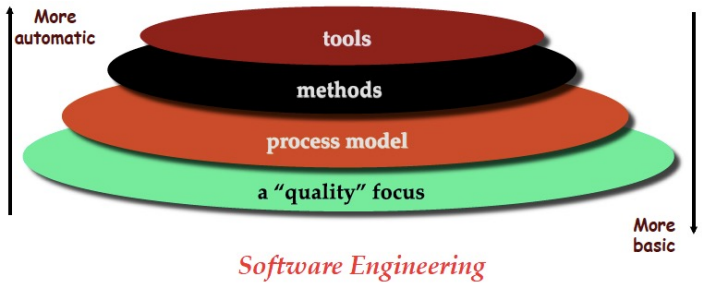
- Software Engineering Code of Ethics
  - Aspirations and details
- Birth of "software engineering"
  - Software crisis
- Definition of *Software Engineering*
  - The application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software
- Leading question for today: *"Is your software of high quality & how do you know?"*

Have you asked…

- Did I write unit tests?
- Were my tests sufficient?
- Can I understand my own code?
- Can somebody else understand my code?
- Can I understand somebody else's code?
- Are there proven solutions to my problem? If so, which one should I choose? What else must I do?
- Should we use GitHub? Should we program in Eclipse?
- What should we do next?
- …

Process Models

- What's a process?
  - A process defines a collection of activities, actions, and tasks that are performed when some work product is to be created.



Software Engineering

 Source: R. Pressman, Software Engineering, 2010

Process & Artifact

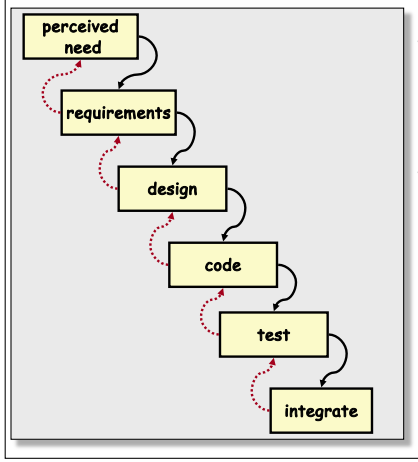
- What's a process?
  - A process defines a collection of activities, actions, and tasks that are performed when some work product is to be created.
- An *artifact* is a piece of information that is produced, modified, or used by a *process*.
  - Code
  - Comments in the code
  - Test
  - Comments in the test
  - Commit message
  - Bug report
  - Feature request
  - Issue status
  - Design mockup
  - …

Key Tenet

Quality(Process) ➔ Quality(Product)

9

Waterfall Model



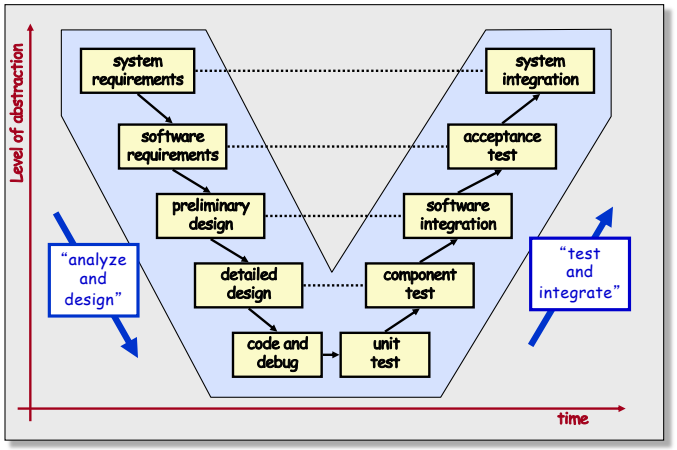
→ View of development:
- a process of stepwise refinement
- largely a high level management view

→ Problems:
- Static view of requirements - ignores volatility
- Lack of user involvement once specification is written
- Unrealistic separation of specification from design
- Doesn't accommodate prototyping, reuse, etc.

 Source: Adapted from Dorfman, 1997, p7 & Loucopoulos & Karakostas, 1995, p29

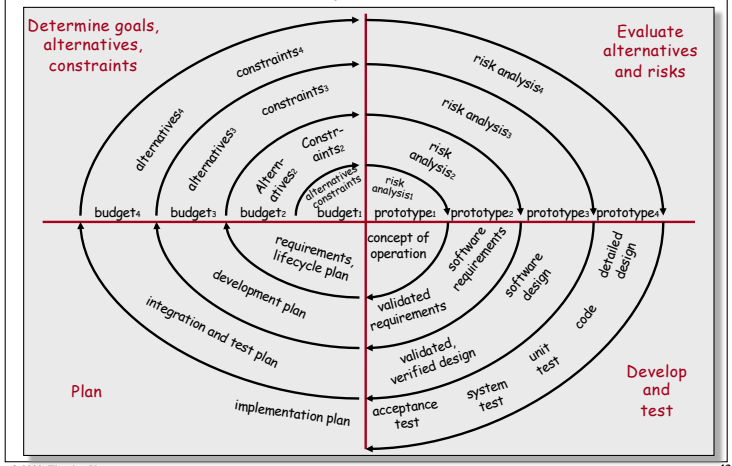
10

V-Model



11

The Spiral Model



 Source: Adapted from Pfleeger, 1998, p57

12

Agile Manifesto

→ In 2001, Kent Beck and 16 other software developers stated
- Individuals and interactions OVER processes and tools
- Working software OVER comprehensive documentation
- Customer collaboration OVER contract negotiation
- Responding to change OVER following a plan

- While there is value in the items on the right, we value the items on the left more

→ Not a process, but a philosophy or set of values

→ What's the work product?
- An operational "software increment" delivered to the customer on the appropriate commitment date

13

Agile Models

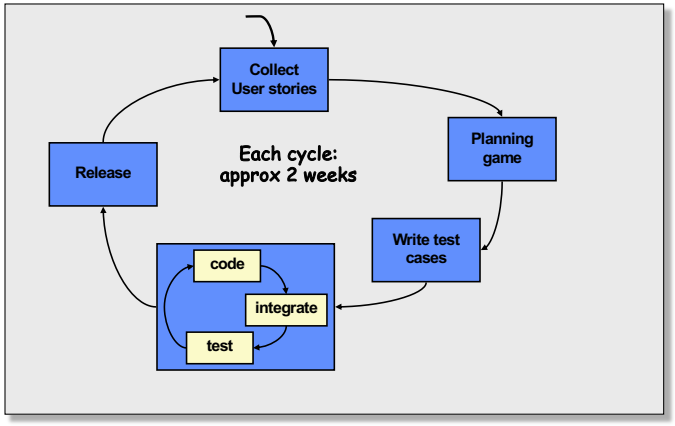
→ Basic Philosophy
- Reduce communication barriers
  - Programmer interacts with customer
- Reduce document-heavy approach
  - Documentation is expensive and of limited use
- Have faith in the people
  - Don't need fancy process models to tell them what to do!
- Respond to the customer
  - Rather than focusing on the contract

→ Weaknesses
- Relies on programmer's memory
  - Code can be hard to maintain
- Relies on oral communication
  - Mis-interpretation possible
- Assumes single customer representative
  - Multiple viewpoints not possible
- Only short term planning
  - No longer term vision

E.g. Extreme Programming
- Instead of a requirements spec, use:
  - User story cards
  - On-site customer representative
- Pair Programming
- Small releases
  - E.g. every three weeks
- Planning game
  - Select and estimate user story cards at the beginning of each release
- Write test cases before code
- The program code is the design doc
  - Can also use CRC cards (Class-Responsibility-Collaboration)
- Continuous Integration
  - Integrate and test several times a day

14

eXtreme Programming (XP)



15

16

Today's Summary

- Quality(Process) ➔ Quality(Product)
  - Artifacts are produced & consumed by a process
  - Waterfall, V-Model, Spiral
  - Agile manifesto & XP

- To-do
  - Review this week's lecture materials
  - Try to finalize your lab team before this Friday (Jan 21)

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
SE History & Ethics
Process Models

This Week:
Intro to OO & Labs

Next Week:
Lab1 Release
Class Diagrams (Cont'd)

2

Software Engineer's Aspirations

→ **Benefits who?**

1. **PUBLIC:** act consistently with the public interest
2. **CLIENT AND EMPLOYER:** consistent with public interest

→ **What to be disciplined about?**

3. **PRODUCT:** meet the highest professional standards possible
4. **JUDGMENT:** maintain integrity and independence
5. **MANAGEMENT:** lead by examples & in an ethical way

→ **Be responsible for whom?**

6. **PROFESSION:** advance the profession's reputation
7. **COLLEAGUES:** be fair and supportive
8. **SELF:** lifelong learning regarding the practice

3

Case Study: To ship or not to ship?

Rachel works as a Quality Assurance Engineer at a large electronics company. She is responsible for the final testing of her company's servers and is part of a team that decides when new products will be shipped to distributors for sale. Lately, the business model requires the release of a new generation of servers approximately every six months, meaning that Rachel has a limited timeframe to conduct her Quality Control tests (that is, performing every possible test on the servers is impossible). Rachel will not ship a product if there is any possibility that the server could malfunction and cause physical harm to the customer. However, she will ship a product that has a higher likelihood of failure resulting in data loss for the customer, because she knows that if she doesn't, her company's competitor will. Is this an ethical way to conduct business?

 Source: https://www.scu.edu/ethics/focus-areas/more/engineering-ethics/engineering-ethics-cases/computer-engineering-cases/

4

Case Study (Cont'd)

1. **PUBLIC:** act consistently with the public interest

   1.04 Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.

2. **CLIENT AND EMPLOYER:** consistent with public interest

   2.06 Identify, document, collect evidence and report to the client or the employer promptly if, in their opinion, a project is likely to fail, to prove too expensive, to violate intellectual property law, or otherwise to be problematic.

3. **PRODUCT:** meet the highest professional standards possible

   3.10 Ensure adequate testing, debugging, and review of software and related documents on which they work.

Labs (40% of your total grade)

→ Team formation information is on Canvas

- Email the instructor (tingting.yu@uc.edu) if you have any questions.
- Contact your teammates if you haven't got to know each other yet.
- Team formation will be finalized by 11:59pm, Friday (Jan 28), i.e., no further change will be allowed after 1/26.

→ Complete the following tasks *as a team* before 11:59pm, Wednesday (Jan 26)

- Create a private, team-wide GitHub repository named, "EECE3093C-SP22-Lab1"
- Invite all your team members, the instructor, and the TAs to the repository

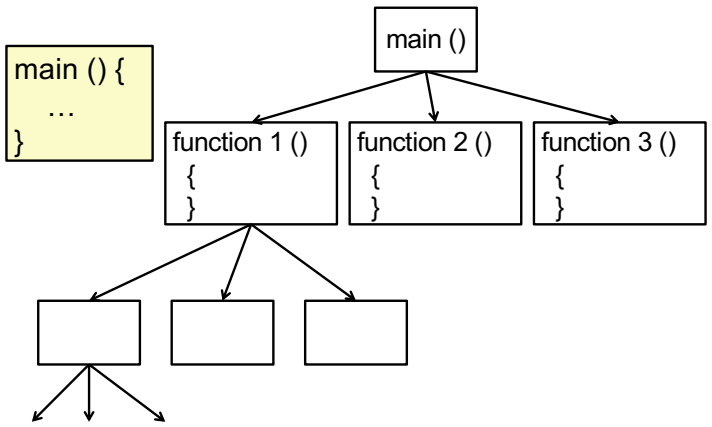
See "05-0126-GitHub.pdf" (7 pages)

How to think about "computation"?

→ Procedural paradigm

- Describe/implement software around the notion of *procedures*
  - e.g., sort, search, cook, run a lecture
- Functional decomposition
  - The system is decomposed into modules
  - Each module is a major function in the application domain
  - Modules can be decomposed into smaller modules

```
main () {
   …
}
```



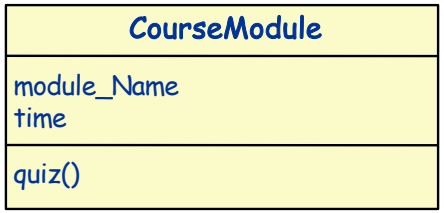
9

Functional Decomposition

- The functionality is spread all over the system
- Maintainer must understand the whole system to make a single change to the system
- Consequence
  - Source code is hard to understand
  - Source code is complex and hard to maintain
  - User interface is often awkward and non-intuitive

10

Object-Oriented (OO) Decomposition:



→ An *object* has state (data) and behavior (function).

→ An *object* is an instance of a class.

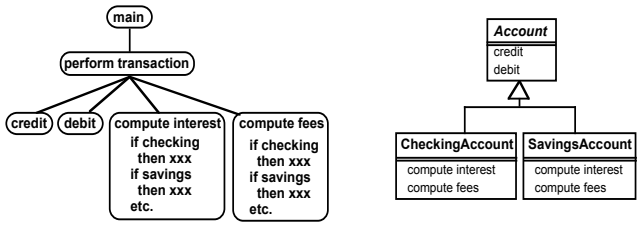
→ A *class* is a blueprint (e.g., initial values of data, implementations of behavior) for creating objects.

11

OO Paradigm

→ An approach to the solution of problems in which all computations are performed in the context of objects

- The objects are instances of classes, which:
  - are data abstractions
  - contain procedural abstractions that operate on the objects
- A running program can be seen as a collection of objects collaborating to perform a given task



 Source: Adapted from Lethbridge/Laganiere, 2001.

12

Quiz #3

- Question #1: Provide your group's GitHub repository link for Lab1?
  - Everybody must know this link & must *answer* this question
  - Remember to invite the instructor and the TA
- Question #2: Do you have any questions about the course (EECE3093C) so far?
  - This question is *optional*, meaning that leaving it unanswered will *not* influence your Quiz3 grade
- Question #3: 1/28 attendance.
  - You shall leave this question *unanswered*, and submit the entire quiz before 11:59pm, Wednesday (1/27)

13

Summary

- Procedural
  - *Doing* things
  - Fundamental to understand computation
- OO
  - Doing *things*
  - More natural to understand application
    - What's an object? What's a class? What're their relationships?
- To-do
  - Complete Quiz3 before Wednesday (Jan 27)
  - Make sure to attend Thursday's (Jan 28) in-person lecture

14

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
SE History & Ethics
Process Models

This Week:
Intro to OO & Labs

Next Week:
Lab1 Release
Class Diagrams (Cont'd)

2

Ice Breakers

→ Q1: OO is an approach to the solution of problems in which all computations are performed in the context of _____.

A. Procedures
B. Classes
C. Functions
D. Objects

→ Q2: An object is _____ of a class.

A. a collection
B. an instance
C. a child
D. a part

3

OO Paradigm

→ An approach to the solution of problems in which all computations are performed in the context of objects

→ Objects are instances of classes

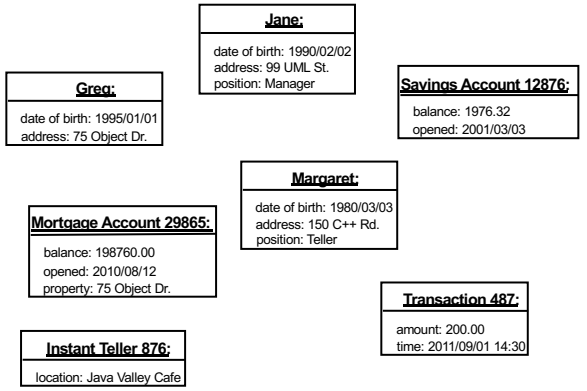
→ Classes are data abstraction containing attributes and operations on the objects

→ A running program can be seen as a collection of objects collaborating to perform a given task

 Source: Adapted from Lethbridge/Laganiere, 2001.

4

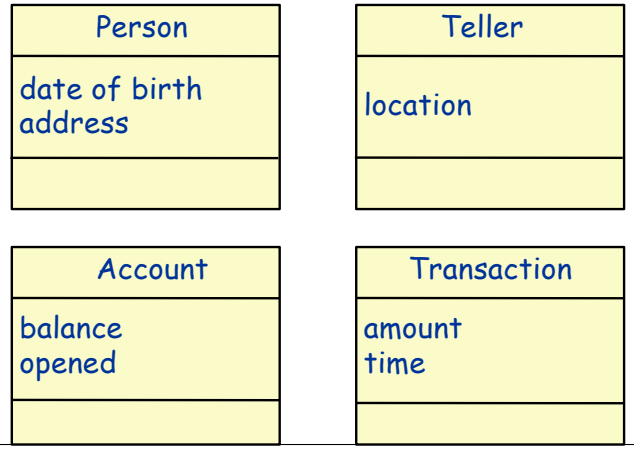
Example: 7 objects created during a running OO program



 Source: Adapted from Lethbridge/Laganiere, 2001.

5

Classes of the above example

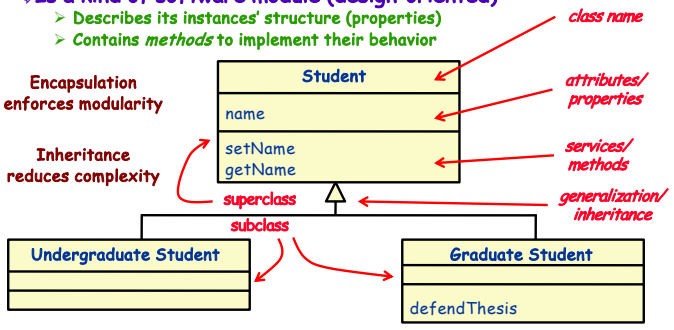


6

Classes

→ A class:

- Is a unit of abstraction in an OO program
- Represents similar objects
  - Objects are instances of a particular class
- Is a kind of software module (design-oriented)
  - Describes its instances' structure (properties)
  - Contains *methods* to implement their behavior

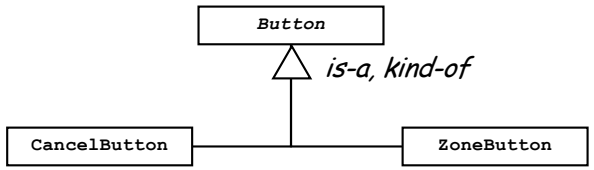


 Source: Adapted from Lethbridge/Laganiere, 2001.

7

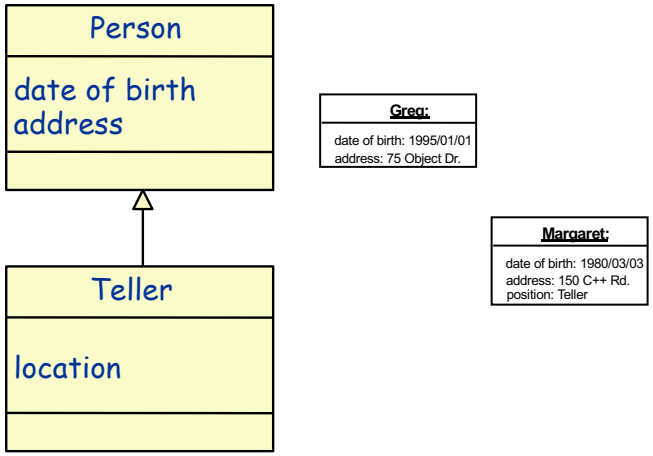
Generalization

- Generalization is implemented via "Inheritance"
- Generalization eliminates redundancy, and supports incrementality



8

Inheritance in action



9

Polymorphism

→ The mechanism by which several methods can have the same name and implement the same abstract operation.

- e.g., the binary operator 1+2 and the binary operation 'a' + 'b' can both inherit from the same superclass "binary operator", but the implementation of + as in 1+2 is different from that of + as in 'a' + 'b'

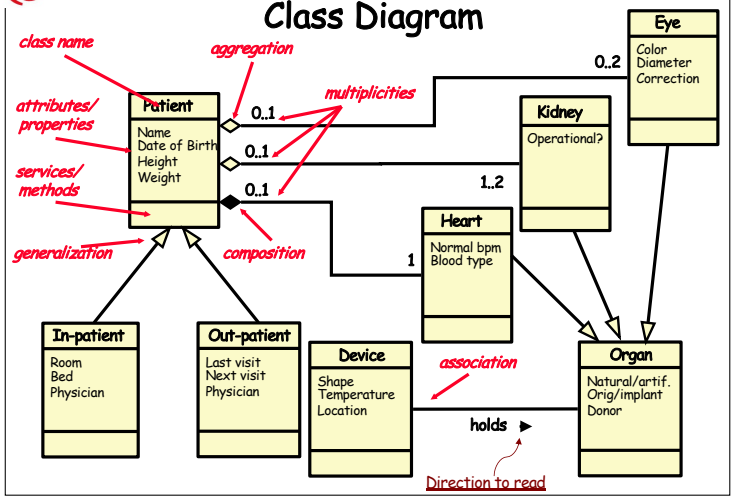
- e.g., a screen cursor may change its shape from an arrow to a hand The routine to move the cursor on screen in response to mouse movement can be written for the "cursor" superclass, but polymorphism lets that cursor take whatever shape it requires at runtime

Test: www.uc.edu

Polymorphism promotes information hiding and therefore flexibility

10

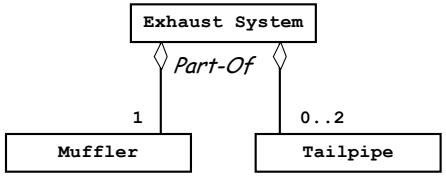
Class Diagram



11

Aggregation

→ An *aggregation* denotes a "consists of" hierarchy

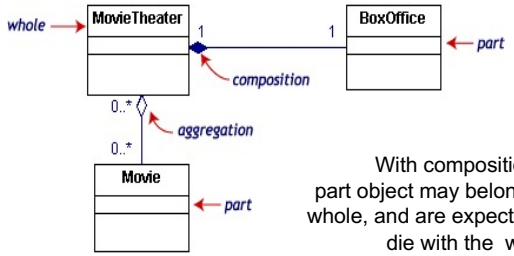


12

Composition

Aggregation ◇

Composition ◆



With composition, the part object may belong to only one whole, and are expected to live and die with the whole.

13

Today's Summary

→ Understanding the OO paradigm & class diagram modeling

- Encapsulation enforces modularity
- Inheritance reduces complexity
- Polymorphism promotes flexibility

→ To-do

- Review this week's lecture materials
- Know your team's Lab1 repo on Github & be ready for Lab1

14

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Intro to OO
Class Diagram Modeling

This Week:
Release Lab1
Class Diagram Modeling (Cont'd)

Next Week:
Creating Objects
JUnit

2

Procedural vs. Object-Oriented

- Procedural
- Object Oriented

Withdraw, deposit, transfer

Customer, money, account

 Source: https://www.alphamatech.com/procedural-vs-object-oriented-programming

3

Lab1

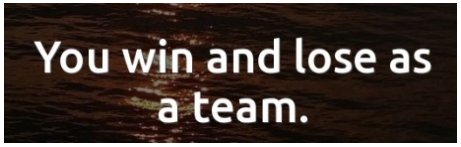
→ Class diagram modeling from procedural code

- Important dates
- TA support
- Inputs
- Task description
- Grading
- Notes

4

Assessments

- Quizzes & synchronous-lecture attendance 40%
- Team-based labs 40%
- Final exam 20%

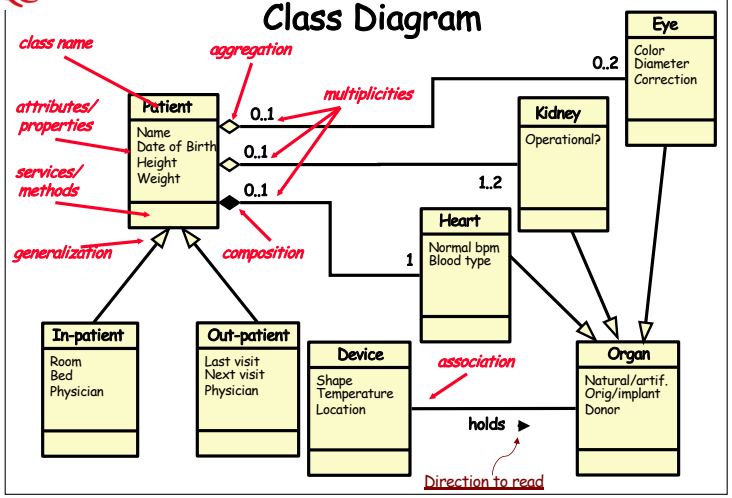


Code of Ethics

IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices

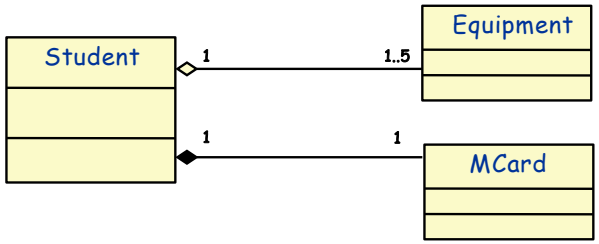
7. COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.

Class Diagram



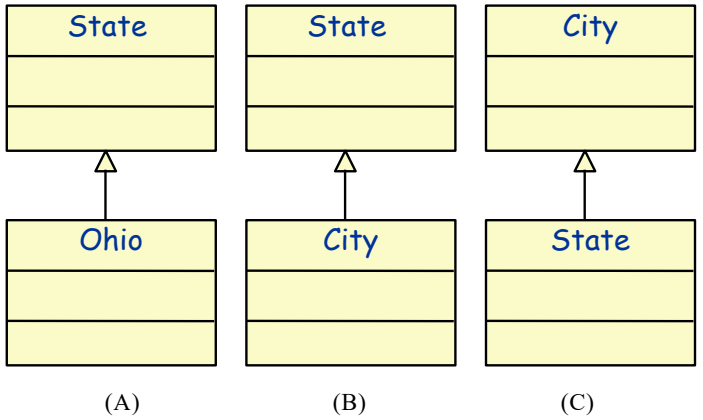
Clarifying *"composition"* and *"aggregation"*

- Java code + separate .txt note

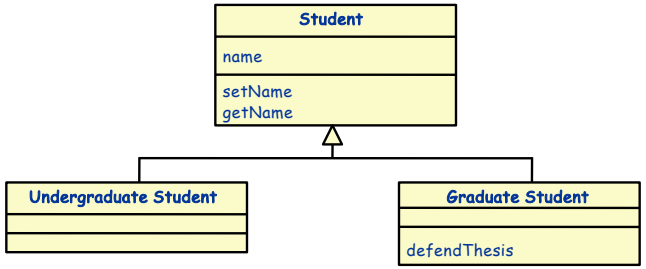


- Conclusion: "composition" and "aggregation" are *different*—both from design and implementation points of view

Understanding "inheritance"



Why isn't "inheritance" having "multiplicity"?



9

Quiz #4

- Question #1: Are (A), (B), and (C) on page 8 correct "inheritance" in your opinion? Why & why not?
  - Everybody must *answer* this question
- Question #2: Do you have any questions about the course (EECE3093C) so far?
  - This question is *optional*, meaning that leaving it unanswered will *not* influence your Quiz4 grade
- Question #3: 2/3 attendance.
  - You shall leave this question *unanswered*, and submit the entire quiz before 11:59pm, Tuesday (2/1)

10

Summary

- Lab1 is released
  - What? When?
  - How?
    - GitHub repo's *Issues* page must NOT stay empty
- Continue learning "class diagram modeling"
  - Clarifying "composition" and "aggregation"
  - Why not "multiplicity" for "inheritance"?
- To-do
  - Begin doing Lab1 *as a team*
  - Complete Quiz4 before Tuesday (Feb 1)
  - Make sure to attend Thursday's (Feb 3) synchronous lecture

11

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

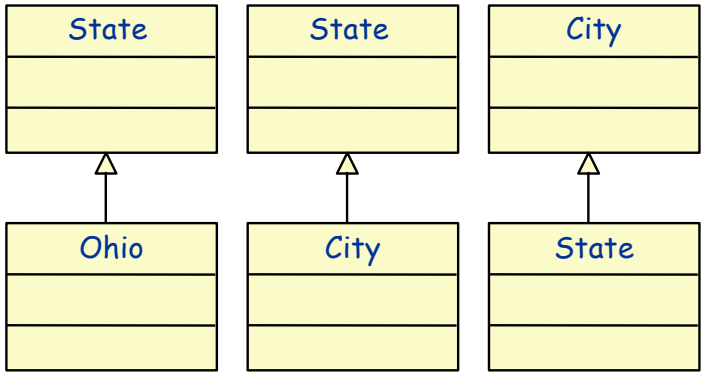
Last Week:
Intro to OO
Class Diagram Modeling

This Week:
Release Lab1
Class Diagram Modeling
(Cont'd)

Next Week:
Creating Objects
JUnit

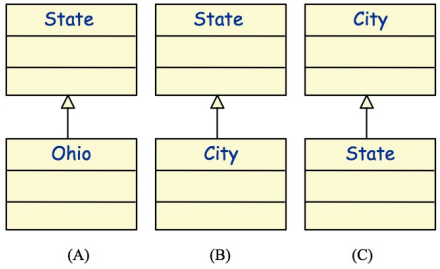
2

Quiz4's Question #1



3

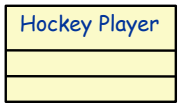
Quiz4's Question #1



None of these are correct!

4

Class Diagram: all and only about…

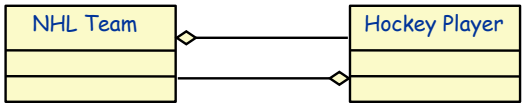


→ From the "Hockey Player" class, one can create 0, 1, 2, … "hockey players"

- Wayne Gretzky
- Nick Foligno

- "Hockey Player" is a *class*
- Wayne Greatzky is an *object*
- Multiplicity is *not* about 1 "Hockey Player" is instantiated to 3 "hockey players", or to 0, 1, 2, …, many "hockey players"

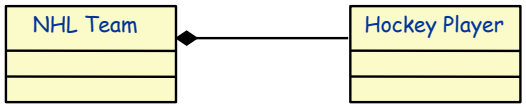
What "has-A" what?



→ In general, modeling "has-A" in both ways is *not* recommended

- "whole" has-A "part"; *not* vice versa
  - A player may be part of a team.
  - A team cannot exist without a player.
  - A player can certainly exist without a team: Wayne Gretzky is *not* part of a team

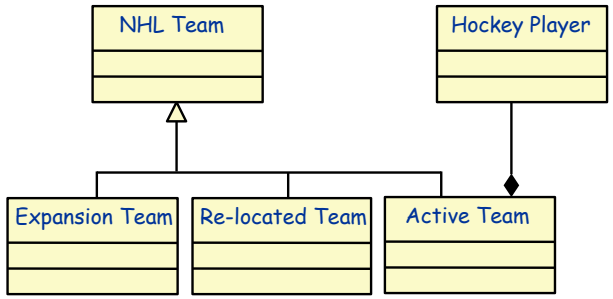
OK, let's use "Composition"



→ Wait a second, what about the Seattle Kraken, Quebec Nordiques…

- Maybe we shall use "Aggregation" instead of "Composition", because it appears that a team can exist without any player
  - Something *fishy* about "Aggregation"

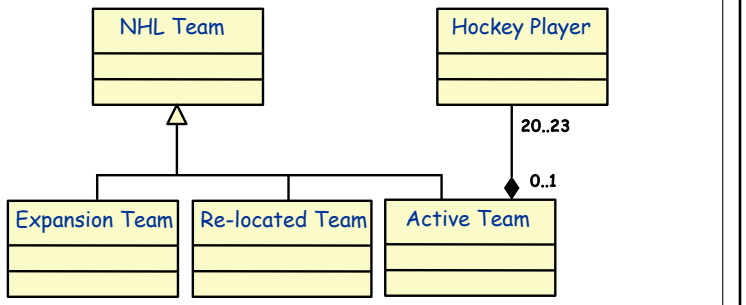
"Inheritance" saves the day



- This process demonstrates "thinking in OO", which makes both problem understanding and solution development more complete, more precise, and more correct

Quality(process) ➔ Quality(product)

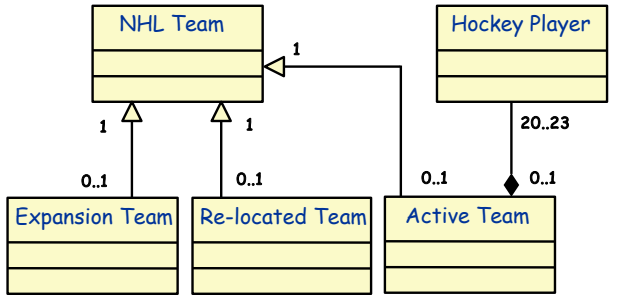
Multiplicity, multiplicity, multiplicity



- Class level: 1 "Active Team" has 20-23 "Hockey Players"
  - Multiplicity quantifies the *amount of participation* in a given class-class relationship
  - There're *four* class-class relationships: inheritance, aggregation, composition, and association

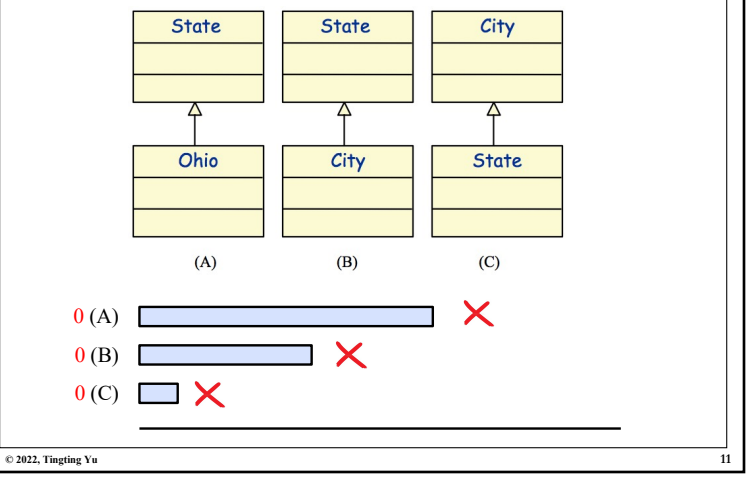
9

Multiplicity of inheritance is always the same



10

The *correct* answers should be…



11

Summary

- Class diagram modeling is about making decisions, even though some decisions will change as software evolves. *Bad decisions* include
  - Incorrect decisions (e.g., having "Aggregation" the wrong way)
  - Confusing decisions (e.g., having "Aggregation" both ways)
  - Weak decisions (e.g., not using "Composition" where it should be used)
  - Always correct decisions (e.g., all "Multiplicity" are having "0..*")
- To-do
  - Continue doing Lab1 *as a team*
    - Milestone (5 points): before 5pm Tuesday (Feb 8), progress percentage & peer evaluation on GitHub repo ("Issues" page)

12

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Class Diagram Modeling

This Week:
Creating Objects
(JUnit)

Next Week:
Design Pattern

2

Lab1 due: 11:59pm, Tuesday (2/15)

→ Make your team's GitHub repo self-contained

↳ Point the TA and me to a "folder" that contains *both* the "class diagram" (e.g., a PNG file) *and* the "notes" to not misunderstand the "class diagram"

→ Make sure to *individually* submit the *completed* "Lab1-SubmissionFile" to Canvas before the end of next Tuesday

3

Lab1 Clarification

→ Do NOT reverse engineer the C code

→ Understand the functionalities provided in the C code

↳ Operands, operators, etc

→ Design a class diagram for these functionalities

→ Think it in OO

4

Quiz5

→ Completion grade & completion grade *no more*

- Answer the quiz, lab, exam questions by considering the *GENERAL* cases
- Pay attention to which Quiz (Question) is completion grade, and which is not

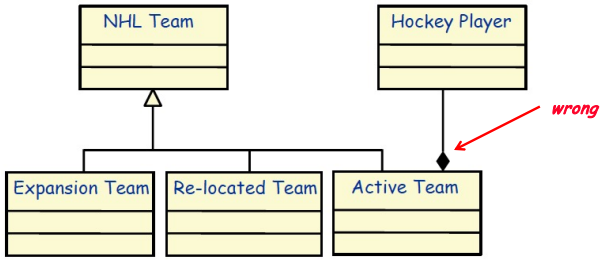
→ Quiz5

- Question 1: "an object" and "an instance" are indistinguishable
- Question 2: In general, *increment()* applies to all the "Numerical" objects, and *isPrime()* applies to only the "Integer" objects
- Question 3: a class can obviously be associated with itself

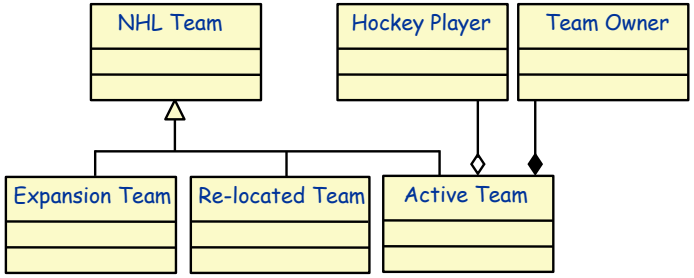
Correction

→ Composition indicates that the existence of the parts depends on the whole.

→ Aggregation indicates that the whole and the part can exist independently.



Composition



"If P, then Q" is the same as "If (not Q), then (not P)".

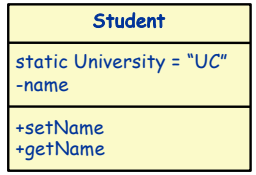
- If the part exists ("Team Owner"), then the whole exists ("Active Team").
- If the whole is deleted ("Active Team"), then the part ("Team Owner") is also deleted as a result.

Members of a Class

→ attributes and methods

→ "static": one unique member bound at compile time

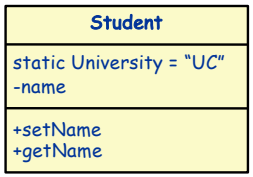
→ *access* to members varies



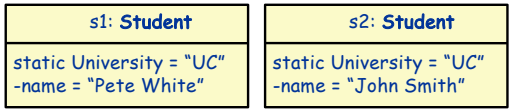
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
no modifier	Y	Y	X	X
private	Y	X	X	X

Class Variable vs. Instance Variable

- "University" is a class variable
  - Due to "static"
  - Shared among *ALL* the objects
- "name" is an instance variable
  - Every instance/object maintains its own "name"



- Demo
  - s2 changes class variable
  - "final": an entity can only be assigned once



Java Access Levels for Members

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
no modifier	Y	Y	X	X
private	Y	X	X	X

- Q1: Where should the constructor be declared?
- Q2: What should the constructor be named?
- Q3: Can a class have more than one constructor?
- Q4: What should be the access level for the constructor(s)?
- Q5: What is the consequence of changing a constructor access level from "public" to "private"?

Summary

- Further clarification about "Composition" and "Aggregation"
- Programming language support offered by Java
  - Access to members of a class: private, no modifier, protected, public
  - "static" defines one unique member and shares that with all objects
  - "final" allows an entity to be assigned only once
- Constructor is made to create objects
  - Where to define a constructor?
  - What name to use for a constructor?
  - How many constructors can a class have?
- To-do
  - Finishing & *submitting* Lab1 *before 11:59pm, Tuesday (2/15)*

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Constructor
(member accessibility, "static", "final")

This Week:
Singleton
JUnit

Next Week:
Quiz #6
Sequence Diagram

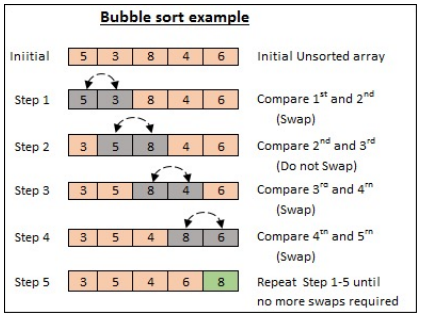
2

What do you think the code does?

```
void function1(int arr[ ]) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++) {
                for (int j = 0; j < n-i-1; j++) {
                        if ( arr[j] > arr[j+1] ) {
                                int temp = arr[j];
                                arr[j] = arr[j+1];
                                arr[j+1] = temp;
                        }
                }
        }
}
```


3

- The code snippet is from https://www.geeksforgeeks.org/bubble-sort/
- The illustrative figure is from https://medium.com/karuna-sehgal/an-introduction-to-bubble-sort-d85273acfcd8



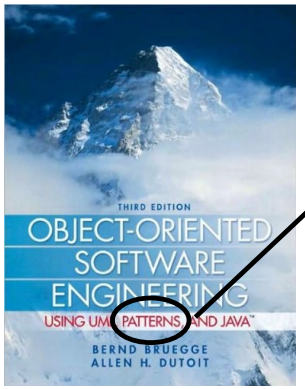
4

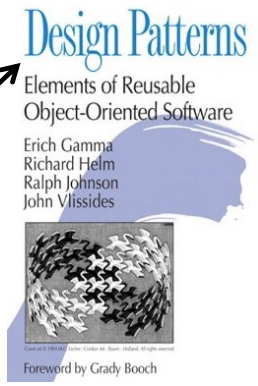
What's good about "Bubble Sort"?

- Reusable
  - Recurring problems
  - Proven solution & its tradeoffs, e.g., $O(n^2)$
- Demo #1:

```
int arr[] = { 64, 11, 90, new Integer('A') };
```


From reusable code to reusable OO design

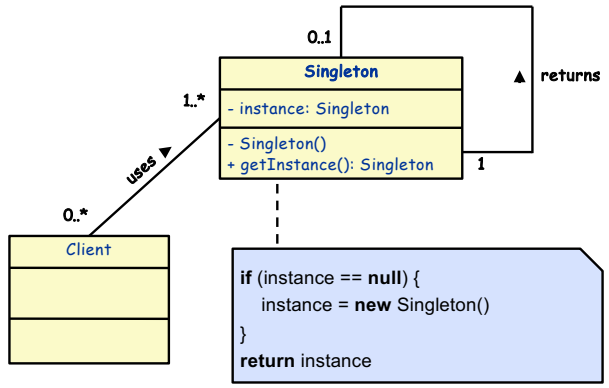




https://refactoring.guru/design-patterns

- How to describe/document a pattern?
  - Intent
  - Problem
  - Solution
  - Real-World Analogy
  - Structure
  - Pseudocode
  - Applicability
  - How to implement
  - Pros and Cons

Singleton Pattern: Structure



Singleton Demo & JUnit

- A *unit testing* framework for Java
  - Kent Beck, Erich Gamma, … https://junit.org
  - Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level
- Annotation
  - @Test, @Before, @After, …
- Assertion
  - assertNull, assertNotNull, assertSame, assertEquals, assertNotEquals, assertTrue, …

Demo

- Singleton.txt

Exercise

```
Student s1 = new Student();
s1.name = "Alice";
Student s2 = s1;
assertEquals(s1, s2);      //1st assertion
Student s3 = new Student();
s3.name = "Emily";
assertEquals(s1, s3);      //2nd assertion
assertSame(s1, s3);        //3rd assertion
s3.name = "Alice";
assertEquals(s1, s3);      //4th assertion
assertSame(s1, s3);        //5th assertion
```


Here're the correct answers

```
Student s1 = new Student();
s1.name = "Alice";
Student s2 = s1;
assertEquals(s1, s2);      //1st assertion      true
Student s3 = new Student();
s3.name = "Emily";
assertEquals(s1, s3);      //2nd assertion      false
assertSame(s1, s3);        //3rd assertion      false
s3.name = "Alice";
assertEquals(s1, s3);      //4th assertion      true
assertSame(s1, s3);        //5th assertion      false
```


Summary

- Patterns are *reusable* OO designs, which are *proven solutions* to a class of *recurring problems*
  - Software *engineering*, instead of software art
  - Building blocks are more *application-oriented*
  - Know (known) tradeoffs (pros & cons)
- Singleton
  - One of the *creational* design patterns
  - Addresses two requirements: ensuring that a class has a single instance & providing a global access point to that instance
- To-do
  - Reviewing Singleton pattern
  - Next Tuesday (2/22): Instructor's Lab1 model answer & Singleton quiz (*NOT* a completion quiz); *no video lecture*
  - Next Thursday (2/24): Video lecture, no attendance

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

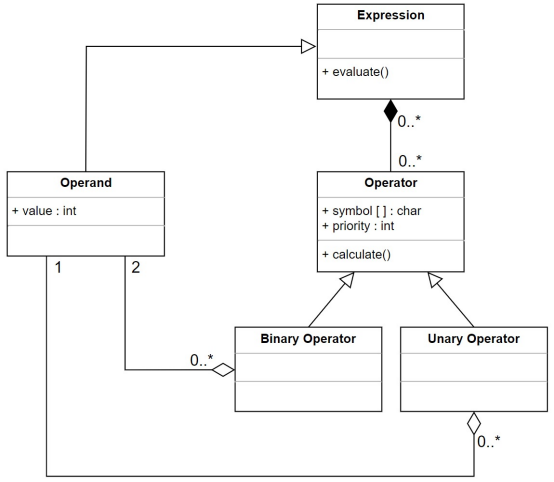
Today's Menu

Last Week:
Singleton
JUnit

This Week:
Lab1 Model Answer
Singleton (Quiz6)
Inheritance & Object Creation

Next Week:
Sequence Diagram
Releasing Lab2

2



3

Quiz 6: Correction

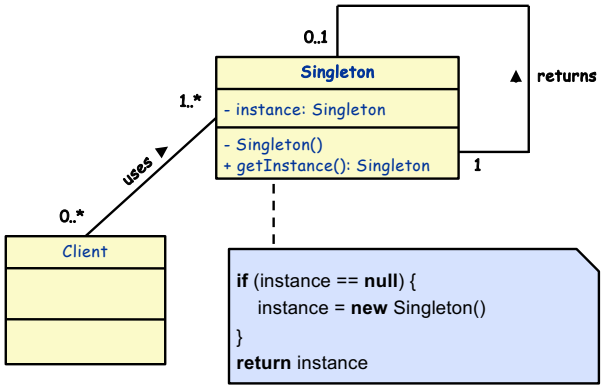
Question 3 1 pts

Referring to page #8 of Thursday's slides (11-0217-singleton.pdf) where the class diagram is shown, which statement is true if the attribute "instance: Singleton" is changed from private to public (i.e., changing the Java code from "private static Singleton instance;" to "public static Singleton instance;")?

- Singleton class no longer has just one instance.
- Besides "getInstance()", a Client will have another global access point to the Singleton instance.
- Neither statements is true.
- Both statements are true.

4

Singleton Pattern: Structure



5

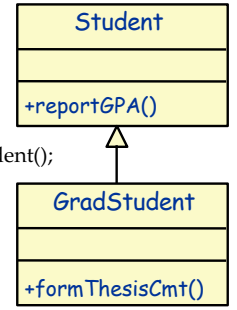
Inheritance in Java

In Java, given "public class Student {…}", its subclass is created by "extends"

public class GradStudent extends Student {…}

Which line (lines) is wrong?

1. Student s1 = new Student();
2. GradStudent gs1 = new GradStudent();
3. s1 = gs1;
4. s1.formThesisCmt();
5. gs1 = s1;
6. gs1.reportGPA();



6

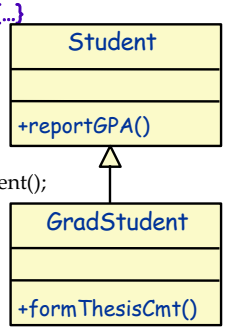
Inheritance in Java

In Java, given "public class Student {…}", its subclass is created by "extends"

public class GradStudent extends Student {…}

Which line (lines) is wrong?

1. Student s1 = new Student();
2. GradStudent gs1 = new GradStudent();
3. ✓ s1 = gs1;
4. ✗ s1.formThesisCmt();
5. ✗ gs1 = s1;
6. ✓ gs1.reportGPA();

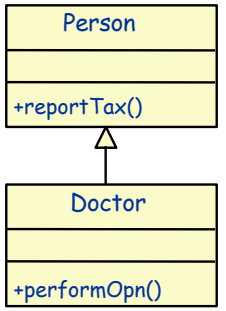


7

Inheritance in Java

```
Person p = new Person();      // p doesn't do anything special
Doctor d = (Doctor) p;        // casting doesn't make p a doctor
d.performOpn();               // desired to FAIL
```

→Design decision: *where* "reportTax()" and "performOpn()" go is important



8

Summary

- Singleton: what's the *implication* if the *pattern* is not followed?
  - "private Singleton()" → "public Singleton()"
  - "private static Singleton instance" → "public static Singleton instance"
  - Singleton is its own client
- Object creation in the face of Inheritance
  - Can a superclass object be cast into a subclass object? If so, can the resulting object perform subclass's method?
  - Can a subclass object be cast into a superclass object? If so, can the resulting object perform superclass's method?
- To-do
  - Java: strongly encouraged
  - As a *team*, be ready for Lab2

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

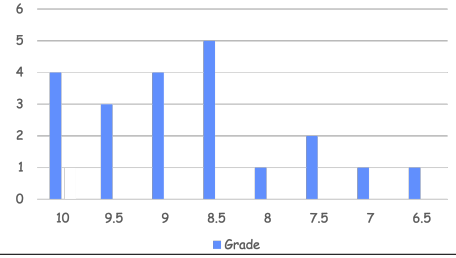
Lab1 graded

→ Keep in mind that:

Your score = Team's score * Individual contribution co-efficient

→ Canvas submission is *mandatory*

→ Email me (tingting.yu@uc.edu) if you have questions



2

Today's Menu

Last Week:
Lab1 Model Answer
Singleton (Quiz6)
Inheritance & Object Creation

This Week:
Sequence Diagram
Releasing Lab2

Next Week:
Behavioral Diagram
Lab2 Q&A

3

Example: 7 objects created during a running OO program

Jane:
date of birth: 1990/02/02
address: 99 UML St.
position: Manager

Greg:
date of birth: 1995/01/01
address: 75 Object Dr.

Savings Account 12876:
balance: 1976.32
opened: 2001/03/03

Margaret:
date of birth: 1980/03/03
address: 150 C++ Rd.
position: Teller

Mortgage Account 29865:
balance: 198760.00
opened: 2010/08/12
property: 75 Object Dr.

Transaction 487:
amount: 200.00
time: 2011/09/01 14:30

Instant Teller 876:
location: Java Valley Cafe

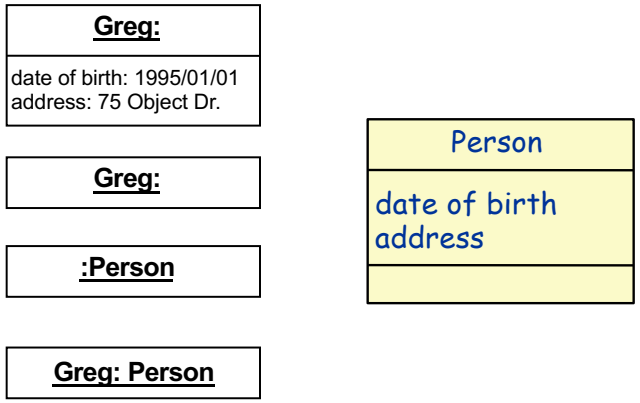
 Source: Adapted from Lethbridge/Laganiere, 2001.

4

Object vs. Class

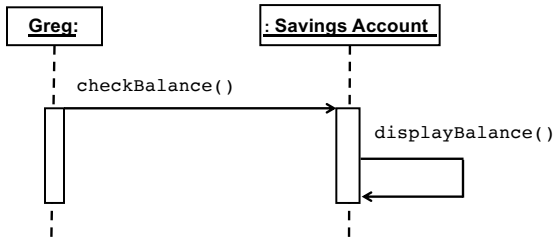
- Object: A single instance
  - e.g., "Greg" is a customer, "Margaret" is a teller
- A class describes a group of objects with similar properties
  - e.g., "Margaret", "Jane" … are bank employees
- Class diagram models a system's structure at the class level
  - What's being encapsulated & how're these encapsulations related to each other?
- Sequence diagram models a system's behavior at the object level
  - How the instantiated objects interact with each other in order to fulfill a requirement (feature)?

Object vs. Class

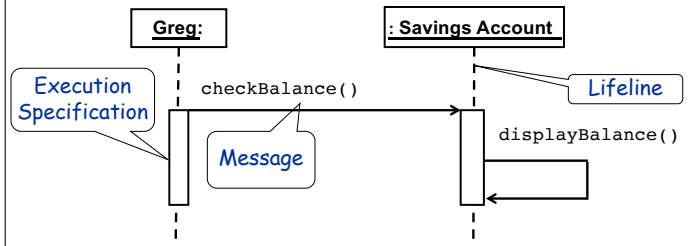


Sequence Diagram

- Horizontally: objects participating in the interaction
  - The reason for the interaction is to fulfill a requirement (e.g., Greg wants to check the balance of his savings account)
- Vertically: time
  - Time always proceeds from top to bottom in sequence diagram

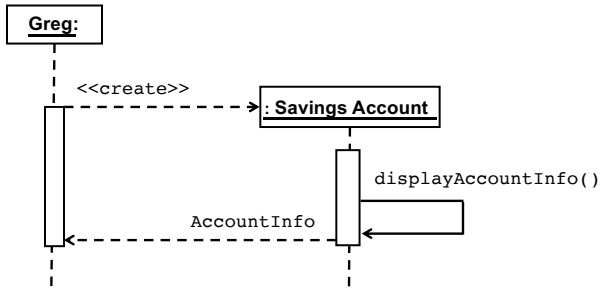


Sequence Diagram: Notations



- A *lifeline* represents *an individual participant* in the interaction.
- An *execution specification* is an occurrence which represents moments in time at which actions or behaviors *start or finish.*
- A *message* captures an exchange between objects.

Lifeline represents the time span when the object can send or receive messages



- Dashed arrows represent *object creation* or *reply/return* (i.e., message results that are sent back to the sender).

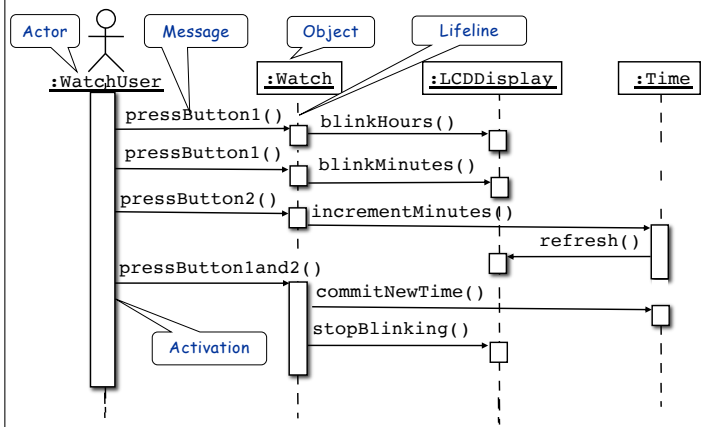
9

Messages

- Define a particular communication between lifelines of an interaction
- Examples of communication
  - Raising a signal
  - Invoking an operation
  - Creating or destroying an instance
- Specify sender and receiver with an arrow

10

Another Sequence Diagram



11

Lab2

- Sequence diagram modeling from Java code
  - Important dates
  - TA support
  - Inputs
  - Task description
  - Grading
  - Notes

12

Summary

→ Sequence diagram

- Modeling a system's behavior (one requirement) at the object level
- Time proceeds from to to bottom
- Objects are interacting horizontally by sending and receiving messages

→ Lab2 is released

→ To-do

- Begin doing Lab2 *as a team*
- TA office hour
- Make sure to attend Thursday's (March 3) in-person lecture

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

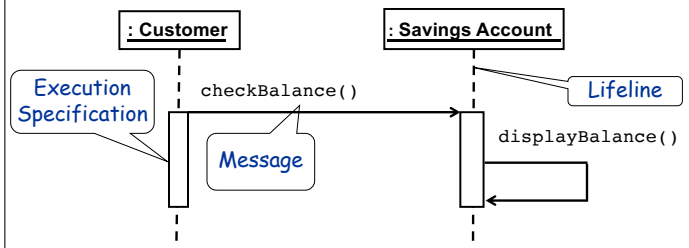
Last Week:
Lab1 Model Answer
Singleton (Quiz6)
Inheritance & Object Creation

This Week:
Sequence Diagram
Releasing Lab2

Next Week:
Behavioral Diagram
Lab2 Q&A

2

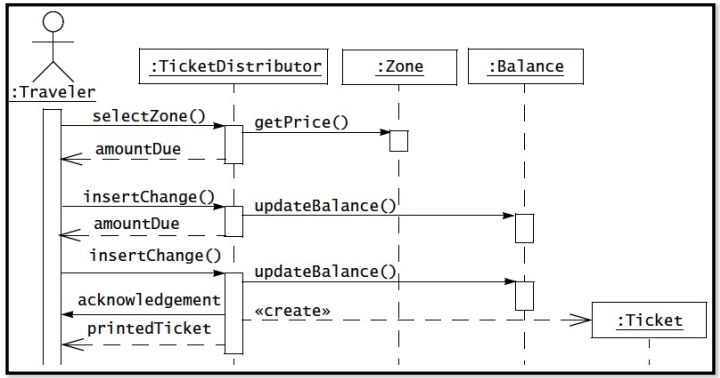
Sequence Diagram: Notations & *Level*



- A *lifeline* represents *an individual participant* in the interaction.
- An *execution specification* is an occurrence which represents moments in time at which actions or behaviors *start or finish.*
- A *message* captures an exchange between objects.

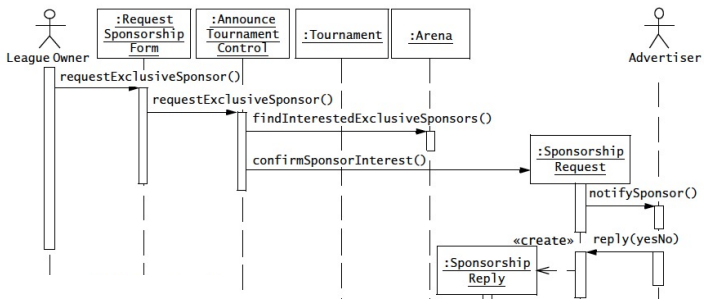
3

Actor can be a specific instance



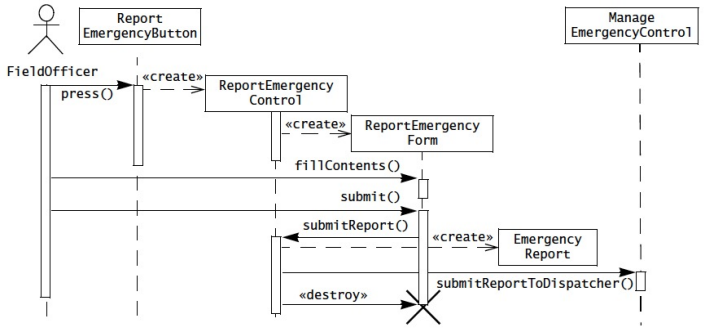
4

Actor can also be an external system



5

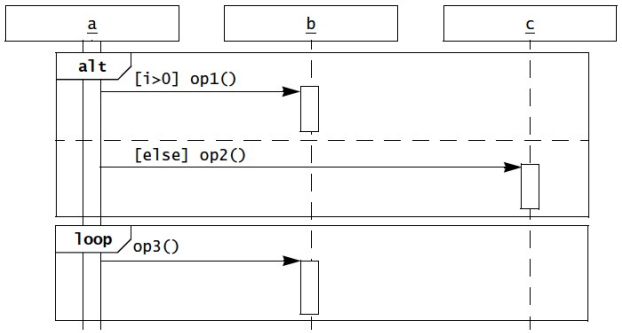
<<create>> and <<destroy>>



- Instances that are *destroyed* during the interaction have a *cross* indicating when the object ceases to exist.
- The object *cannot* send or receive messages below the cross sign.

6

Conditions and iterations



- For example, "inserting a valid bank card" or "inserting an invalid bank card".
- Once the condition is met, "at most three attempts to enter a valid password"

7

Lab2 Notes

- Modeling choices (e.g., actor)
- Tuesday (3/8) before 5pm
  - Milestone check: completion grade
  - Make sure to use the same Lab1 Git Repo
- Final submission – everybody – before 11:59pm, Tuesday (3/22)
- Questions?

8

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

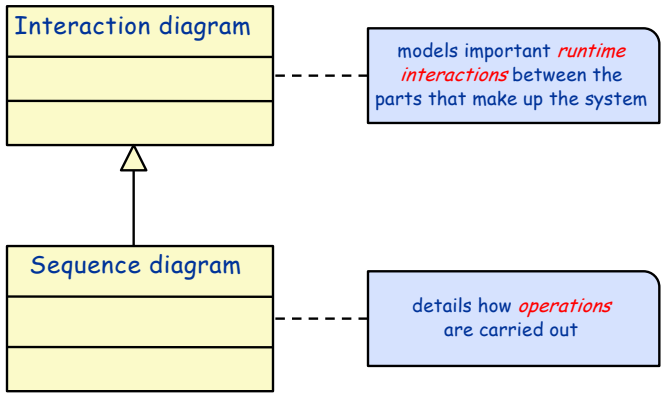
Last Week:
Sequence Diagram

This Week:
Working on Lab2
Sequence Diagram (Q&A)

After Spring Break:
Behavioral Diagram

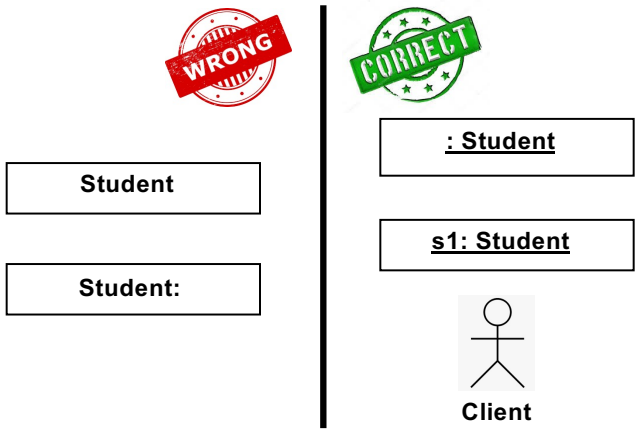
2

Behavioral vs. Structural



3

Naming the objects



4

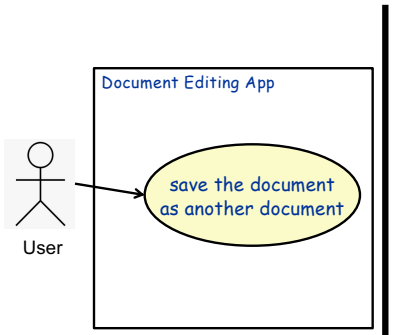
When & why to use sequence diagram?

- → Design phase & NOT implementation phase
- → Model *high-level interaction* between active objects in a system
  - Not low-level implementations
- → Model the interaction between objects (instances) within a collaboration that *realizes a use case*
  - Not sequential, conditional, or iterative code blocks
- → Capture *high-level interaction* between user of the system and the system, between the system and other systems, or between subsystems
  - Not low-level (RPI) calls, socket communications, etc.

 Source: http://www.inf.ed.ac.uk/teaching/courses/seoc/2008_2009/notes/LectureNotes08_notes.pdf

5

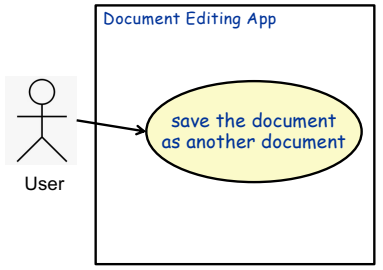
UML (design) vs. Java (implementation)




```
...
doc.saveAS(name,
      type_list);
...
```


6

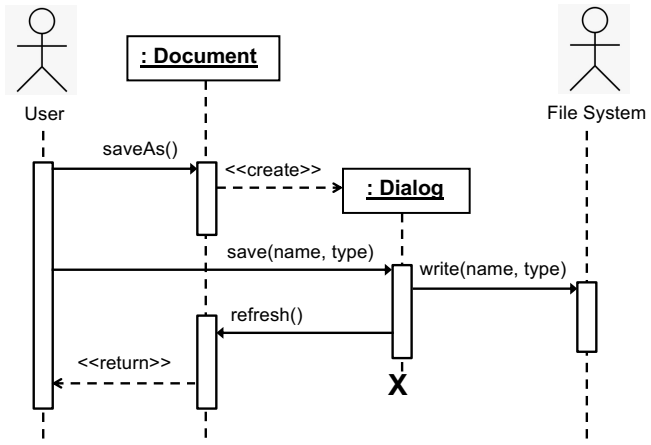
Java code changes, but design doesn't




```
...
type_list = "doc"
      | "docx"
      | "txt";
doc.saveAS(name,
      type_list);
...
```

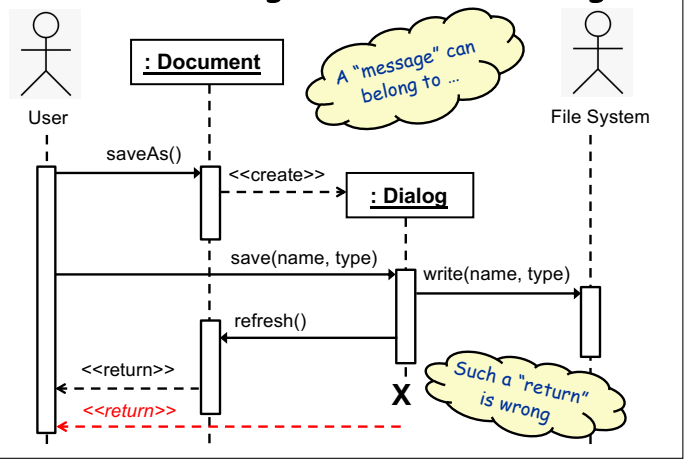

7

Detailing how "saveAs()" is carried out



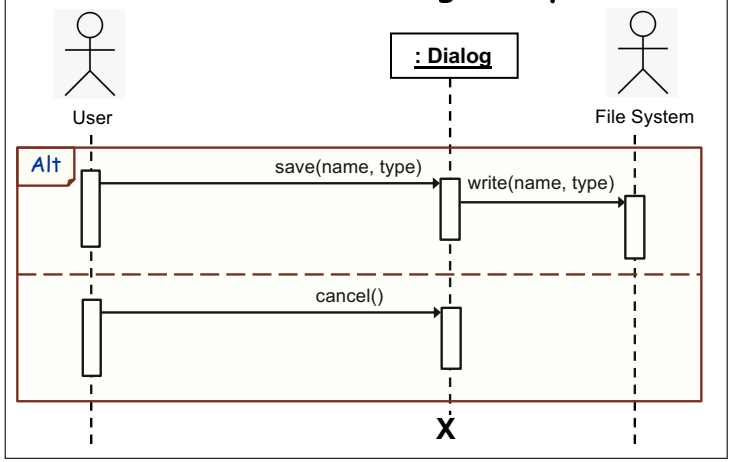
8

Does NOT change if Java code changes



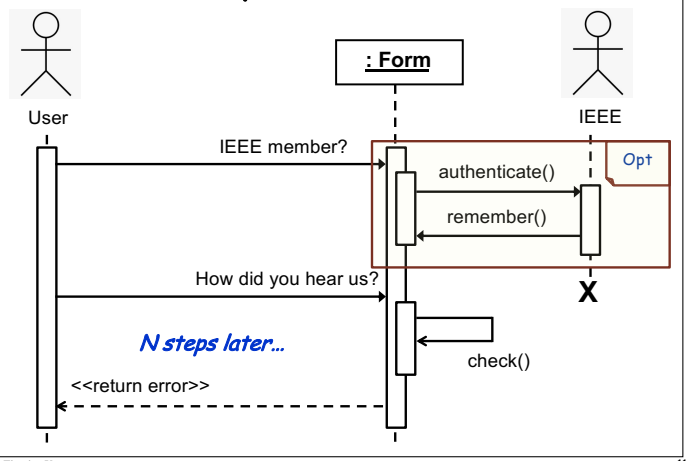
9

"Alt" to continue detailing the operations



10

Why is it valuable?



11

https://conf.researchr.org/attending/icse-2021/registration

Dear Nan,

I choose "Main Conference Registration Only" but when I press the "Apply" button on the last step of ICSE'21 registration still I get the following message: "The Discount Code you have entered is not applicable with your selection."
Attached you can find a snapshot of my screen showing the error.
Please let me know if you need more information.

Thank you for your help.

Regards,
[redacted]

12

Summary

- Sequence diagram is about *high-level interaction*
  - Why do objects, sub-systems, and/or external systems interact with each other *is more important than* how the interaction is implemented in Java
  - An object, no matter how it is being created (one long line of code, or five lines of code), is created *for a purpose*, and in sequence diagram, the purpose is to *interact* with others to fulfill a requirement
- To-do
  - Finish & *submit* Lab2 before 11:59pm, Tuesday (3/22/2022)

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week: Sequence Diagram (Cont'd)

This Week: State Machine Diagram

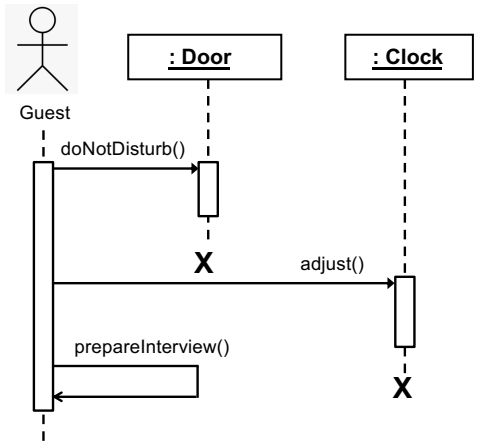
Next Week: Use Cases & Req.s Eng.

2

Scenario

3

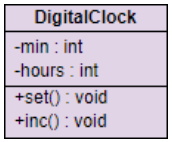
Sequence diagram: high-level interactions

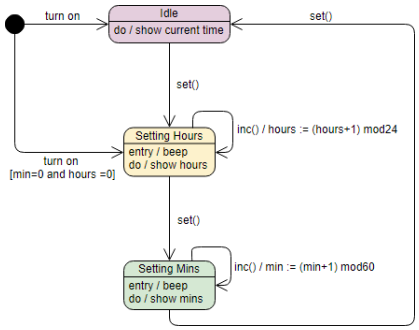


4

A state machine diagram (SMD)

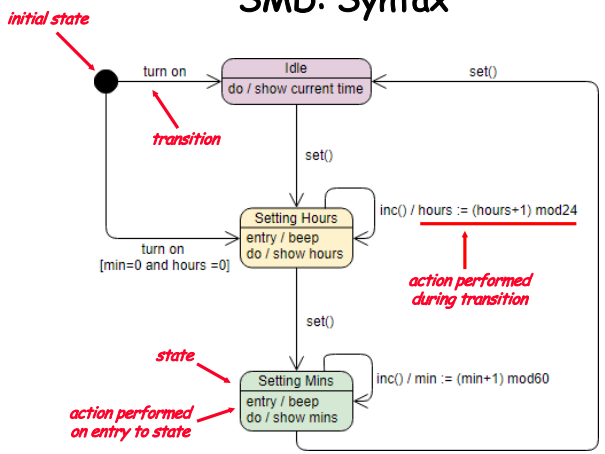
→ used to model the *behavior* of individual objects, use cases, or entire systems





5

SMD: Syntax



6

SMD: Notes

→ Not only is there a *finite* set of states, but a state is to be in for a *finite* amount of time.

→ A transition maps "current state" x "event" x "condition" → "next state", and sometimes also performs "action" during the transition

- "initial state" x "turn on" → "Idle"
- "initial state" x "turn on" x "[min=0 and hour=0]" → "Setting Hours"
- "Setting Hours" x "inc()" → "Setting Hours" x "hours := (hours+1) mod 24"

→ This clock does *not* turn off, does *not* dec() hour or min, does *not* change hour & min together, does *not* change hour immediately after changing min …

7

Quiz 7

→ Due before 11:59pm, Wednesday (3/23/2022)

→ *Completion* grade

→ Describe, in your own way, the states of the SMD.

→ Describe, in your own way, the transitions of the SMD.

→ Describe, in your own way, the entire SMD.

8

Summary

- Sequence diagram is about *high-level interaction*
  - Why do objects, sub-systems, and/or external systems interact with each other *is more important than* how the interaction is implemented in Java
  - An object, no matter how it is being created (one long line of code, or five lines of code), is created *for a purpose*, and in sequence diagram, the purpose is to *interact* with others to fulfill a requirement
- To-do
  - Finish & *submit* Lab2 before 11:59pm, Tuesday (3/22/2022)

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Sequence Diagram (Cont'd)

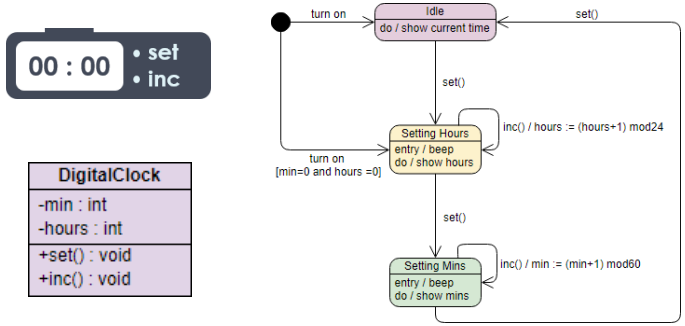
This Week:
State Machine Diagram

Next Week:
Requirements and User Cases

2

A state machine diagram (SMD)

→ used to model the *behavior* of individual objects, use cases, or entire systems



 Source: https://online.visual-paradigm.com/diagrams/tutorials/state-machine-diagram-tutorial/

3

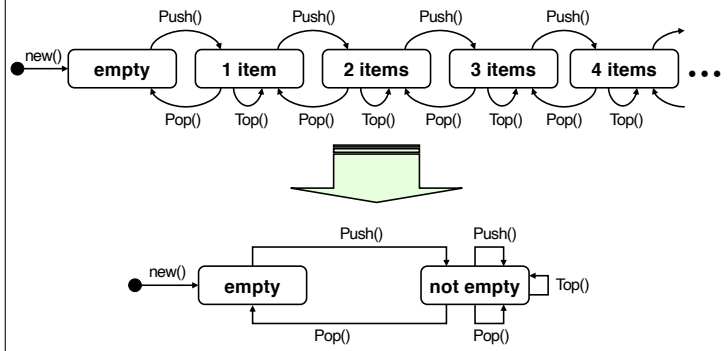
Abstraction

→ The state space of most objects is enormous
- State space size is the product of the range of each attribute
  - E.g. object with 3 boolean attributes: 2^3+1 states
  - E.g. object with 2 integer attributes: |Dom(var1)| x |Dom(var2)| +1 states
  - E.g. object with 1 real-valued attributes: …?
- If we ignore computer representation limits, the state space is infinite

→ Only part of that state space is "interesting"
- Some states are not reachable
- Integer and real values usually only vary within some relevant range
- We're usually not interested in the actual values, just certain ranges:
  - E.g. for Age, we may be interested in age<18; 18≤age≤65; and age>65
  - E.g. for Cost, we may only be interested in cost≤budget, cost=0, cost>budget, and cost>(budget+10%)

4

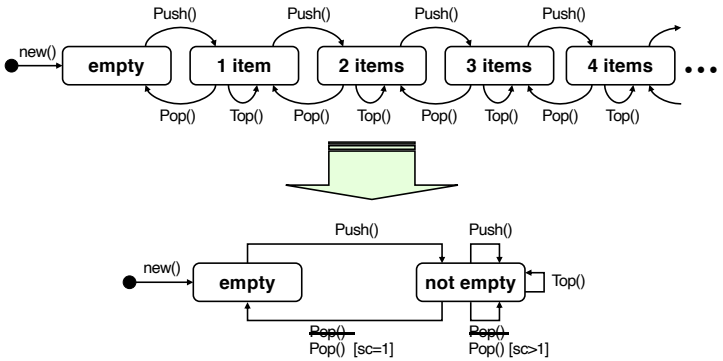
Collapsing The State Space



- The abstraction usually permits more traces
  - E.g. this model does not prevent traces with more pops than pushes
  - But it still says something useful

5

Collapsing The State Space



- The abstraction usually permits more traces
  - E.g. this model does not prevent traces with more pops than pushes
  - But it still says something useful

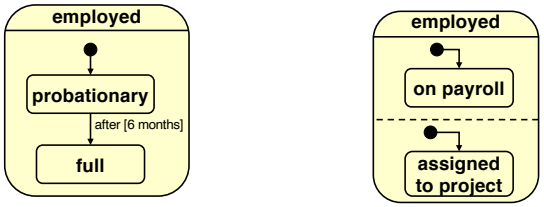
6

Superstates of Statecharts [Harel' 87]

→States can be nested, to make diagrams simpler
- A superstate consists of one or more states.
- Superstates make it possible to view a state diagram at different levels of abstraction.

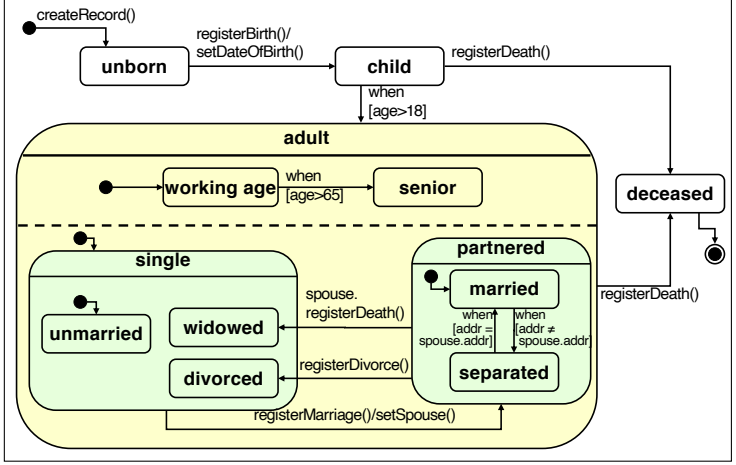
→ OR superstates
- when the superstate is "on", only one of its substates is "on"

→ AND superstates (concurrent substates)
- When the superstate is "on", all of its states are also "on"
- Usually, the AND substates will be nested further as OR superstates



7

Hierarchical Statecharts



8

Checking Your Statecharts

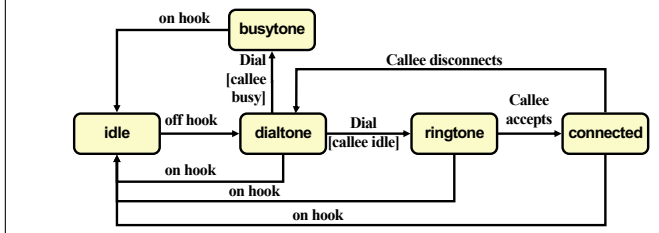
→ Consistency Checks
- All events in a statechart should appear as:
  - operations of an appropriate class in the class diagram and
  - incoming messages for this object on a collaboration/sequence diagram
- All actions in a statechart should appear as:
  - operations of an appropriate class in the class diagram and
  - outgoing messages for this object on a collaboration/sequence diagram

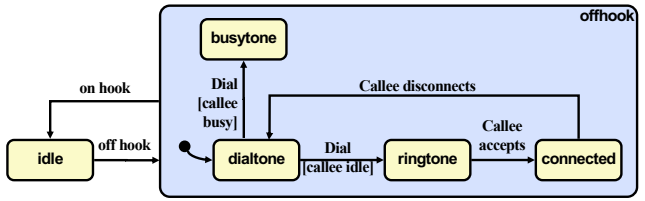
→ Style Guidelines
- Give each state a unique, meaningful name
- Only use superstates when the state behaviour is genuinely complex
- Do not show too much detail on a single statechart
- Use guard conditions carefully to ensure statechart is unambiguous
  - Statecharts should be deterministic (unless there is a good reason)

→ You probably shouldn't be using statecharts if:
- you find that most transitions are fired "when the state completes"
- many of the trigger events are sent from the object to itself
- your states do not correspond to the attribute assignments of the class

9

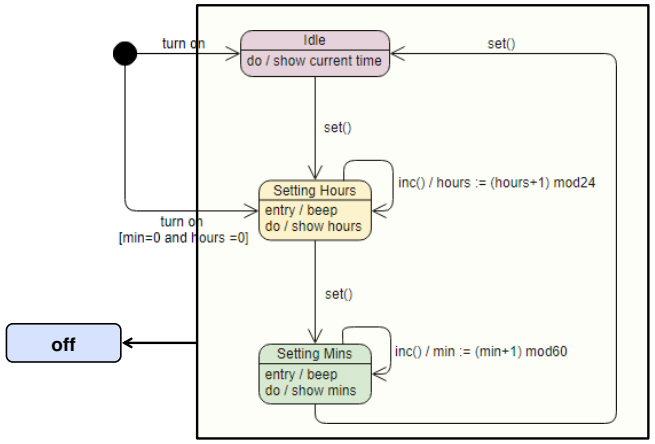
Refresher: FSMs and Statecharts





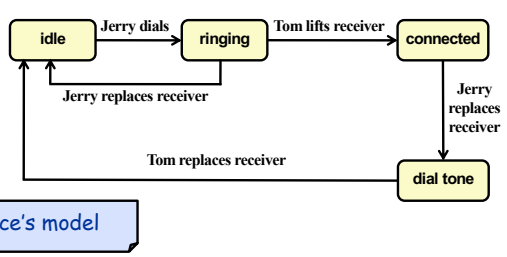
10

Add an "off" state



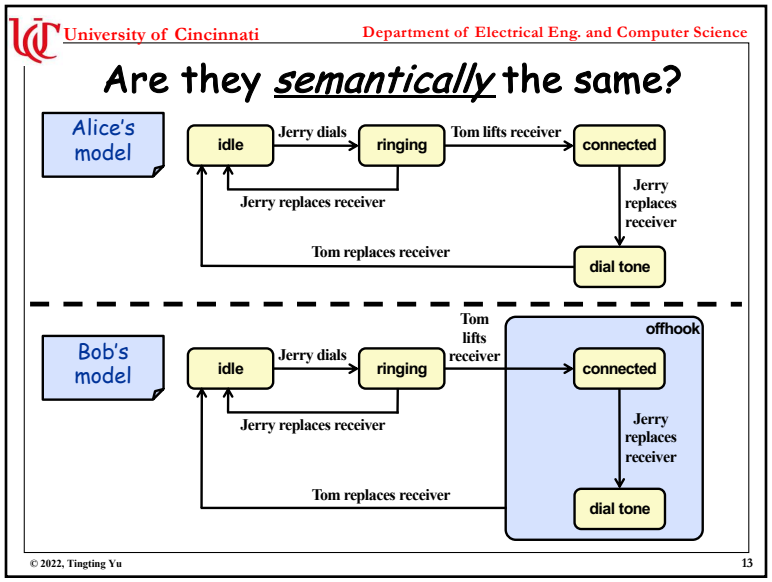
11

Is syntax sugar always sweet?

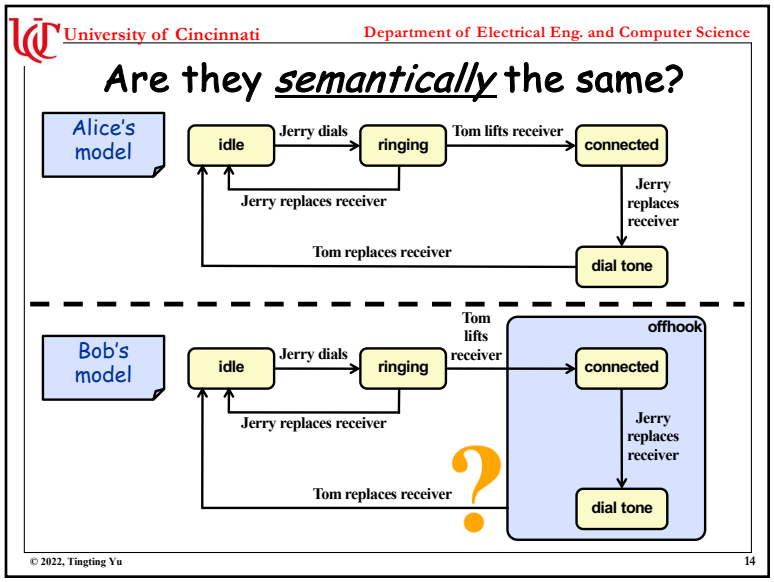


Alice's model

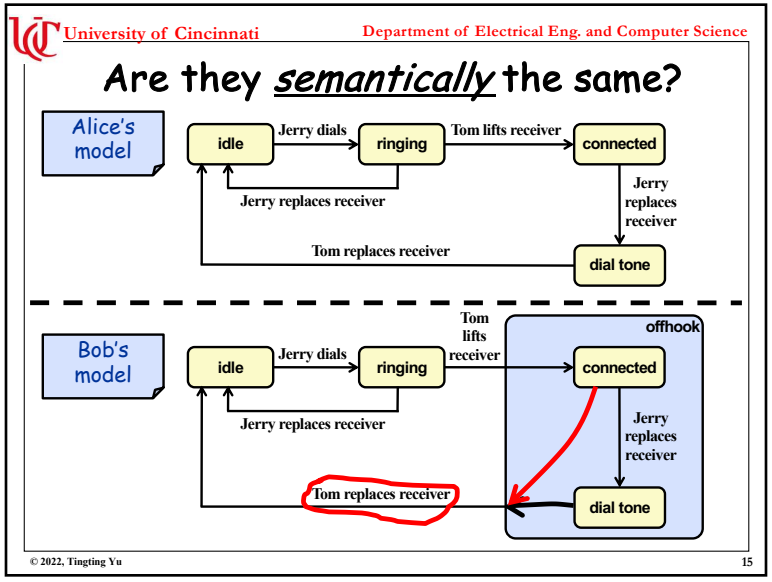
12


University of Cincinnati
Department of Electrical Eng. and Computer Science
Are they semantically the same?
Alice's model
idle
Jerry dials
ringing
Tom lifts receiver
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
Bob's model
idle
Jerry dials
ringing
Tom lifts receiver
offhook
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
© 2022, Tingting Yu
13

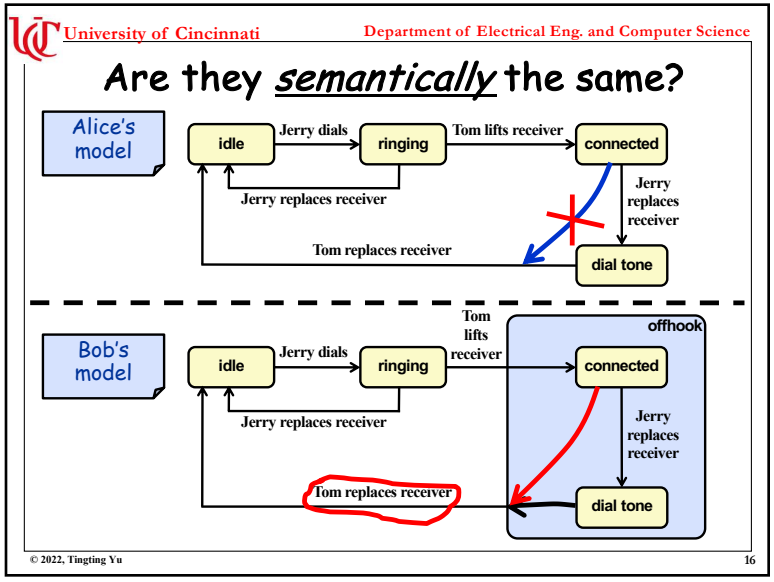
13


University of Cincinnati
Department of Electrical Eng. and Computer Science
Are they semantically the same?
Alice's model
idle
Jerry dials
ringing
Tom lifts receiver
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
Bob's model
idle
Jerry dials
ringing
Tom lifts receiver
offhook
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
?
© 2022, Tingting Yu
14

14


University of Cincinnati
Department of Electrical Eng. and Computer Science
Are they semantically the same?
Alice's model
idle
Jerry dials
ringing
Tom lifts receiver
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
Bob's model
idle
Jerry dials
ringing
Tom lifts receiver
offhook
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
© 2022, Tingting Yu
15

15

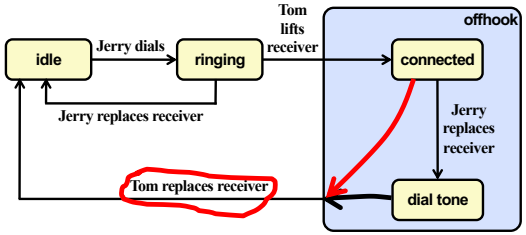

University of Cincinnati
Department of Electrical Eng. and Computer Science
Are they semantically the same?
Alice's model
idle
Jerry dials
ringing
Tom lifts receiver
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
Bob's model
idle
Jerry dials
ringing
Tom lifts receiver
offhook
connected
Jerry replaces receiver
Jerry replaces receiver
Tom replaces receiver
dial tone
© 2022, Tingting Yu
16

16

Are they *semantically* the same?

Alice's model

Bob's model



17

Quiz 8

- Due before 11:59pm, Tuesday (3/29/2022)
- *Completion* grade
- Describe, in your own way, the states of the SMD.
- Describe, in your own way, the transitions of the SMD.
- Describe, in your own way, the entire SMD. .
- Comparison of Quiz 7

19

Summary

- SMD can be used to model the behavior of an object, a sub-system, or the entire system
  - Abstraction, superstate, parallelism
  - Useful for uncovering (requirements) incompleteness and inconsistency
- To-do
  - Complete Quiz 8 before 11:59pm, Tuesday (3/29/2021)
  - Next week
    - Use Cases
    - Lab 3

20

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

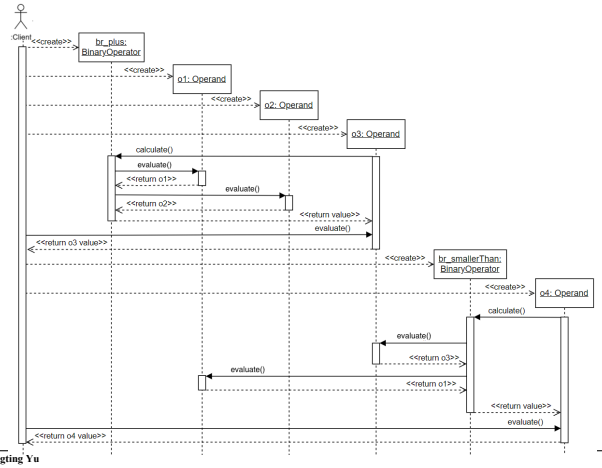
Last Week:
State Machine Diagram (Cont'd)

This Week:
Use Case & Lab3

Next Week:
OO wrap up

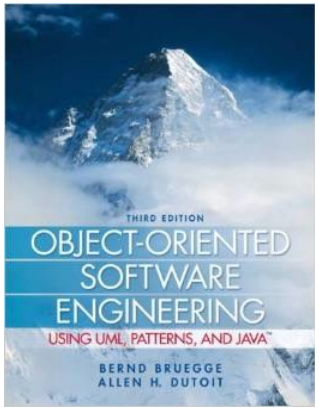
2

Lab2 model answer



3

The three pillars of this course



Source: *www.amazon.com*

4

UML (Unified Modeling Language)

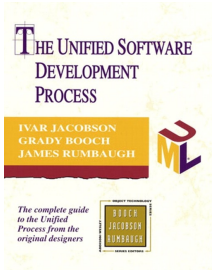
- Nonproprietary standard for modeling SW systems
  - Managed by OMG (Object Mgmt Group)
  - Current version: UML 2.2
    - http://www.uml.org/
  - Commercial tools
    - Rational, Together, Visual Architect…
  - Open-source tools
    - ArgoUML, StarUMML, Umbrello …
- Designed by committee
  - Use case diagrams
  - Class diagrams
  - Message sequence charts (sequence diagrams)
  - Activity diagrams
  - State Diagrams (uses Harel's statecharts)
  - Module Diagrams
  - …

OBJECT MANAGEMENT GROUP

omg.org

Why "unifying"?

- Convergence of different notations used in OO methods (late 1980s-early 1990s), mainly
  - OMT (James Rumbaugh & colleagues), OOSE (Ivar Jacobson), & Booch (Grady Booch)
- They also developed the Rational Unified Process '99



Why "unifying"?

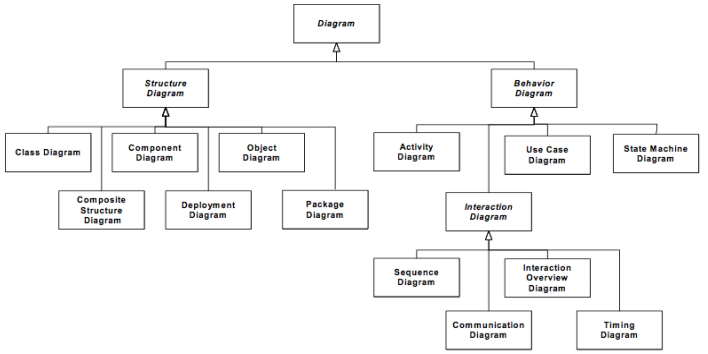
- Convergence of different notations used in OO methods (late 1980s-early 1990s), mainly
  - OMT (James Rumbaugh & colleagues), OOSE (Ivar Jacobson), & Booch (Grady Booch)
- They also developed the Rational Unified Process '99

25 year at GE Research, where he developed OMT, joined (IBM) Rational in 1994, CASE tool OMTool

At Ericsson until 1994, developed use cases and the CASE tool Objectory, joined IBM Rational since 1995

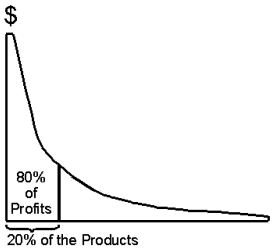
Developed the Booch method ("clouds"), ACM Fellow 1995, and IBM Fellow 2003

Another View on UML Diagrams



Pareto Principle (80-20 rule)

- You can solve 80% of the modeling problems by using 20% UML
- The 20% that we learn
  - Class diagrams
  - Sequence diagrams
  - Use case diagrams
  - State machine diagrams



Vilfredo Pareto, 1848-1923
Introduced the concept of Pareto Efficiency,
Founder of the field of microeconomics.

9

Our Learning of the UML

- Use case diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Classes, attributes, methods, inheritance, aggregation, composition, association
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- State machine diagrams
  - Describe the dynamic behavior of an individual object, sub-system, or entire system

10

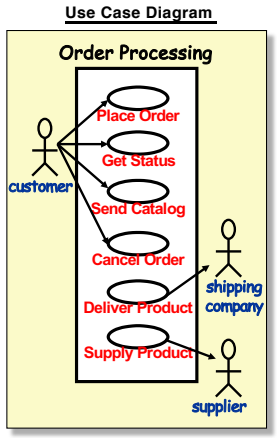
Use Cases

- What is a use case?
  - Each different way that an actor interacts with a system is a use case
    - "a description of a *sequence of actions* that a system performs that yields an *observable result of value to a particular actor*" [Booch]
    - All the use cases need to be enumerated (or the requirements will not be complete)
  - A description of a set of possible scenarios, with a common purpose
  - Typically written in natural language
  - No internal description of the system implementation; just the interaction
- Combining use cases
  - UC1 "uses" UC2
    - the process of doing UC1 always involves doing UC2 at least once
    - a brief, mnemonic way to think about the "uses" arrow: "UC1 has a UC2"
  - UC1 "extends" UC2
    - the process of UC1 is a special case behavior of the same type as the more general process of UC2
- Advantages & Disadvantages
  - detailed characterization of all possible interaction with the system
  - helps in drawing system boundary, and scoping the requirements
  - don't confuse use cases with a precise specification

 Source: Adapted from Rumbaugh 1997, p123-124

11

Use Cases - Example



Use case diagrams represent the functionality of the system from user's point of view

An **actor** is a model for an external entity which interacts (communicates) with the system (software-to-be)

- Role, e.g., "customer", "admin"
- External system, e.g., "CC auth.", "GPS"
- Physical environment, e.g., "weather"

System boundary of the software-to-be

- Scoping the problem & solution

Classifier of the software-to-be, e.g., "Order Processing" in the example

13

Use Case (UC), e.g., Place Order

- A UC represents a class of functionality provided by (& visible from the outside of) the system
- A UC can be described textually, with a focus on the event flow between actor and system
- The textual UC description consists of 6 parts
  1. Unique name
  2. Participating actors
  3. Entry conditions (i.e., preconditions)
  4. Exit conditions (i.e., postconditions)
  5. Flow of events
  6. Special requirements (e.g., exceptions)

14

Use Case Description - Example

Use Case Description

Name: Place Order

Precondition: A valid user has logged into the system.

Description:

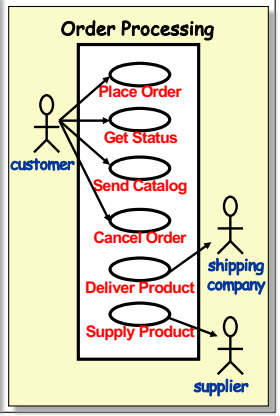
1. The use case starts when the customer selects Place Order.
2. The customer enters his or her name and address.
3. If the customer enters only the zip code, the system will supply the city & state.
4. The customer will enter product codes for the desired products.
5. The system will supply a product description and price for each item.
6. The system will keep a running total of items ordered as they are entered.
7. The customer will enter credit card payment information.
8. The customer will select Submit.
9. The system will verify the information, save the order as pending, and forward payment information to the accounting system.
10. When payment is confirmed, the order is marked Confirmed, an order ID is returned to the customer, and the use case ends.

Exceptions:

In step 9, if any information is incorrect, the system will prompt the customer to correct the information.

Postcondition: The order has been saved in the system and marked confirmed.

Use Case Diagram



15

IEEE Standard for SRS

Source: Adapted from IEEE-STD-830-1993 ***See also****, Blum 1992, p160*

1 Introduction Purpose Scope Definitions, acronyms, abbreviations Reference documents Overview	Identifies the product, & application domain Describes contents and structure of the remainder of the SRS
2 Overall Description Product perspective Product functions User characteristics Constraints Assumptions and Dependencies	Describes all external interfaces: system, user, hardware, software; also operations and site adaptation, and hardware constraints Summary of major functions Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)
3 Specific Requirements Appendices Index	All the requirements go in here (i.e. this is the body of the document). IEEE STD provides 8 different templates for this section

16

Summary

- UML is designed by committee
  - Use cases model a software system's functional requirements from a user's perspective
  - Boundary is of crucial importance
  - Use case actors do *not* always have to be people
- To-do
  - Keep in mind that
    - Quiz 8 is due (Tuesday, March 29)
    - Lab 3 is released (Tuesday, March 29)

19

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Lab3 released

This Week:
OO wrap up

Next Week:
Software Testing

2

Q1: What's the problem for the following code?

```
public WeblogPermission getWeblogPermission(Weblog weblog, User user) throws WebloggerException {
    TypedQuery<WeblogPermission> q = strategy.getNamedQuery("WeblogPermission.getByUserName&WeblogId"
            , WeblogPermission.class);
    q.setParameter(1, user.getUserName());
    q.setParameter(2, weblog.getHandle());
    try {
        return q.getSingleResult();
    } catch (NoResultException ignored) {
        return null;
    }
}

public WeblogPermission getWeblogPermissionIncludingPending(Weblog weblog, User user) throws WebloggerException {
    TypedQuery<WeblogPermission> q = strategy.getNamedQuery("WeblogPermission.getByUserName&WeblogIdIncludingPending",
            WeblogPermission.class);
    q.setParameter(1, user.getUserName());
    q.setParameter(2, weblog.getHandle());
    try {
        return q.getSingleResult();
    } catch (NoResultException ignored) {
        return null;
    }
}
```


3

Q2: What's the problem for the following code?

```
getOrder().getCustomer().getAddress().getState()
```


4

University of Cincinnati Department of Electrical Eng. and Computer Science

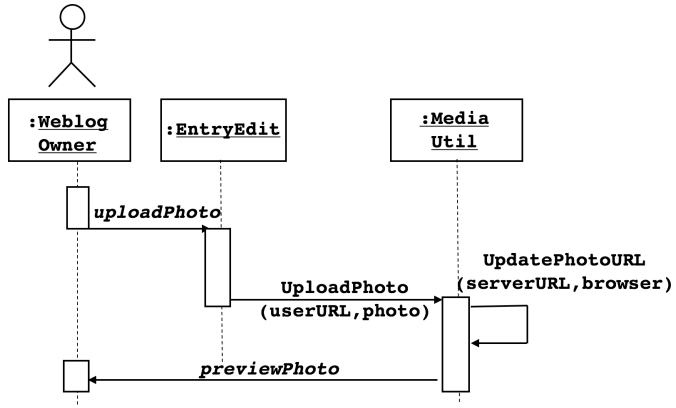
Feature: upload a photo to blog

PREVIEW

5

University of Cincinnati Department of Electrical Eng. and Computer Science

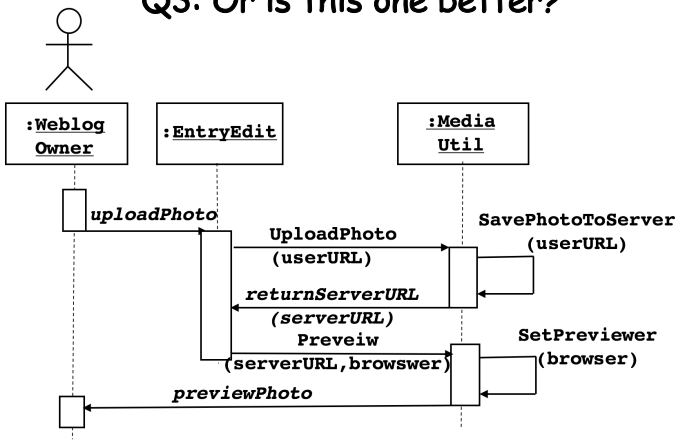
Q3: Is the following design better?



6

University of Cincinnati Department of Electrical Eng. and Computer Science

Q3: Or is this one better?



7

University of Cincinnati Department of Electrical Eng. and Computer Science

Reminder

- TA office hour
  - Wednesdays and Fridays
- Thursday's in-person lecture
  - Wrapping up OO
- Lab3 milestone check (5 points)
  - Documenting your team's progress on the "Issues" page of your GitHub repo, *before 5pm, Tuesday (4/12/2021)*

8

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Lab3 released

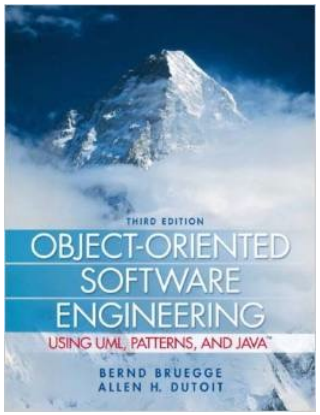
Submission

This Week:
OO wrap up: principles

Next Week:
Software Testing
Lab3 milestone check
5pm, Tuesday (4/12)

2

The three pillars of this course



Source: *www.amazon.com*

3

OO introduced earlier in Jan

→ Paradigm shift

- Compared with procedural paradigm, emphasizes more on data abstraction

→ Software (program & design) is centered around

- Objects (=instances of class)

→ Key constructs & *principles*

- Encapsulation enforces modularity
- Inheritance reduces complexity
- Polymorphism promotes flexibility

4

Q1: What's the problem for the following code?

```
public WeblogPermission getWeblogPermission(Weblog weblog, User user) throws WebloggerException {
    TypedQuery<WeblogPermission> q = strategy.getNamedQuery("WeblogPermission.getByUserName&WeblogId"
            , WeblogPermission.class);
    q.setParameter(1, user.getUserName());
    q.setParameter(2, weblog.getHandle());
    try {
        return q.getSingleResult();
    } catch (NoResultException ignored) {
        return null;
    }
}

public WeblogPermission getWeblogPermissionIncludingPending(Weblog weblog, User user) throws WebloggerException {
    TypedQuery<WeblogPermission> q = strategy.getNamedQuery("WeblogPermission.getByUserName&WeblogIdIncludingPending",
            WeblogPermission.class);
    q.setParameter(1, user.getUserName());
    q.setParameter(2, weblog.getHandle());
    try {
        return q.getSingleResult();
    } catch (NoResultException ignored) {
        return null;
    }
}
```


5

Keep it DRY

DRY (Don't Repeat Yourself)

Every piece of knowledge must have a single, unambiguous, and authoritative representation within a system.

8

Q2: What's the problem for the following code?

```
getOrder().getCustomer().getAddress().getState()
```

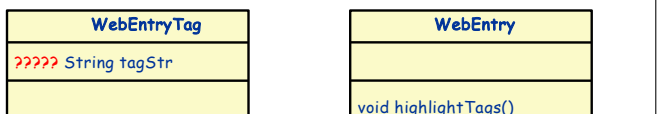

9

Keep it shy

The best code is very shy. Like a four-year old hiding behind a mother's skirt, code shouldn't reveal too much of itself and shouldn't be too nosy into others affairs.

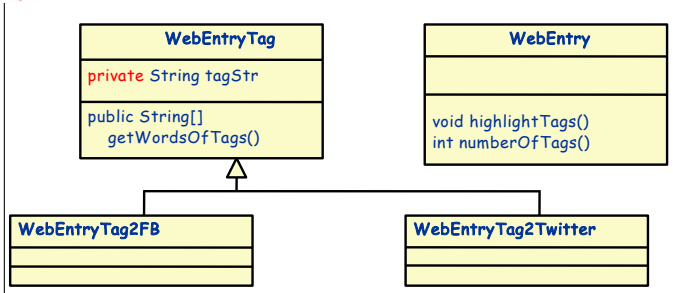
```
getOrder().getCustomer().getAddress().getState()
```


10



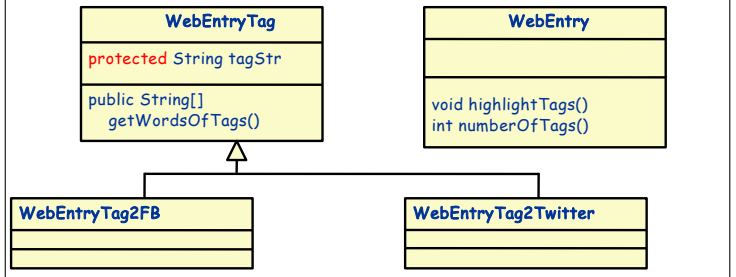
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
no modifier	Y	Y	X	X
private	Y	X	X	X

11



Displaying #Jordan or @Jordan depends on the context of tags, e.g., "Michael Jordan", "Jordan, Asia", etc.

14



15

Keep it shy

The best code is very shy. Like a four-year old hiding behind a mother's skirt, code shouldn't reveal too much of itself and shouldn't be too nosy into others affairs.

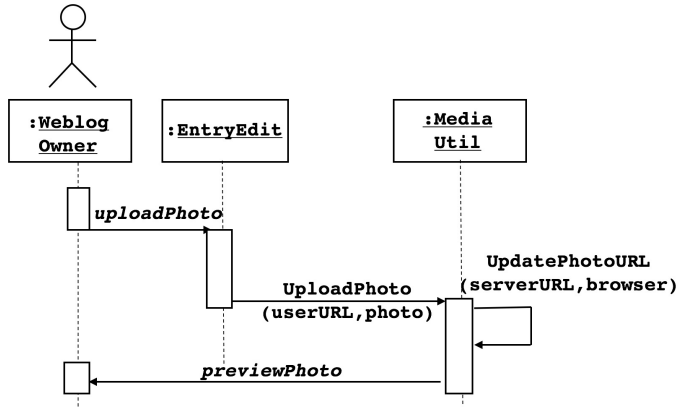
```
getOrder().getCustomer().getAddress().getState()
```


16

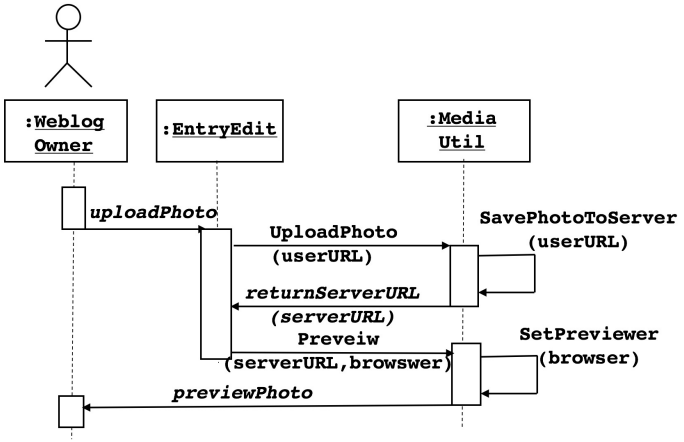
Feature: upload a photo to blog

PREVIEW

17



18



19

Tell the other guy

One of our favorite OO principles is "Tell, Don't Ask"

21

On OO

Some people feel that "getting" object-oriented programming is a difficult, time-consuming process. But does it need to be that hard? And is the difficulty even specific to OO programming?

22

OO in One Sentence:

Keep It DRY, Shy, and Tell the Other Guy

So remember to "keep it DRY, keep it shy, and tell the other guy." SM

Link to the above article can be found in "Canvas" → "Quick links to items"

23

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Keep it DRY, Shy, and
Tell the Other Guy

Submission

This Week:
Lab 3 Checkpoint
Software Testing

Next Week:
Software Laws
Course Summary
Lab 3 due

2

Last lecture (Thursday, 4/21)

→ Lab3 presentation (will be drawn randomly)

↳ Team & then member within that team

→ Summary & final exam preview

3

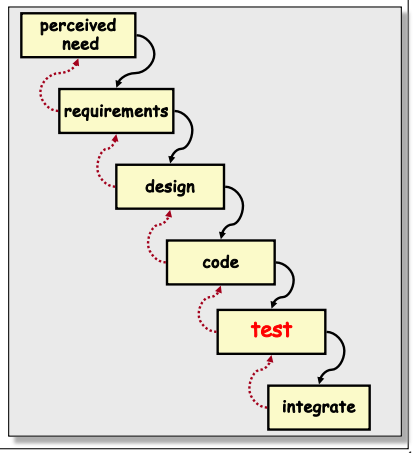
Software Testing

What?

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

How?

A "run-observe-followup" cycle (particularly in case of failure observations)



4

Practically

- How many tests are needed?
- What principle(s) to use in order to write them?

Testing Cannot Prove "no faults"

- Testing can't show correctness (unless we use all possible inputs.)
- We can't use all inputs, so which do we use and when do
- How long does it take (approximately) to test exhaustively the following program?

```
int sum (int a, int b) {return a + b; }
```

- $2^{32} * 2^{32} = \sim 2^{64}$ tests
- Assume 1 test per nanosecond (10^9 tests per second)
  ~2777777 hours = ~115740 days = ~321 years
- methodologies for testing software.

What is a test?

- Run program with known inputs (test inputs/data), check results (w/ test oracles)
  - Tests pass or fail

Testing Effort

- Reported to be >50% of development cost [e.g., Beizer 1990]
- Microsoft: 75% time spent testing
  - 50% testers who spend all time testing
  - 50% developers who spend half time testing

Correctness

- Common (partial) properties
  - Segfaults, uncaught exceptions
  - Resource leaks
  - Data races, deadlocks
- Specific properties
  - Requirements
  - Specification

9

Soundness and Completeness

- Do we detect all bugs? (Completeness)
  - Impossible for dynamic analysis
- Are reported bugs real bugs? (Soundness)
  - Easy for dynamic analysis
- Most practical techniques and tools are both unsound and incomplete!
  - False positives
  - False negatives

10

Testing "Phases"

- Unit (module) -- individual "units"
- Integration – groups of units
- System – software with hardware
- Acceptance – am I satisfied?
- Regression – after changes
- Beta – at specific user sites

11

xUnit testing tools

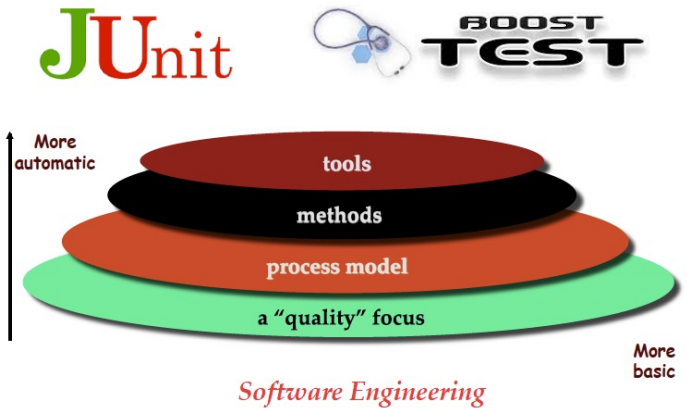
- Programmer's testing tools
- Automated testing!
- Test-first design

☐Each code unit requires several tests

1-12

12

Software testing methods and tools



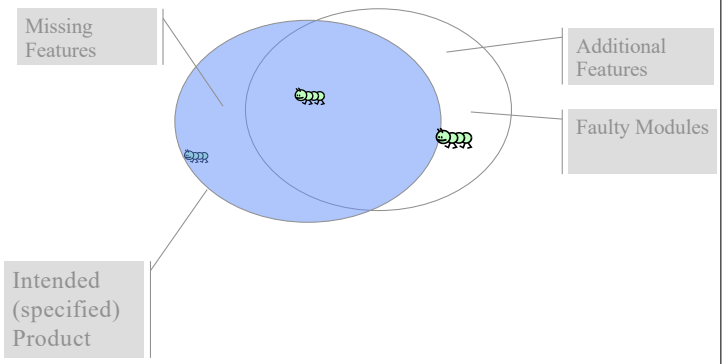
13

Two Classes of Adequacy Criteria: Black Box and White Box

- Black Box (aka Spec-Based, aka Functional)
  - Tests derived from functional requirements
    - Input/Output Driven
  - Internal nature of the software is not relevant to design the tests
- White Box (aka Code-Based, aka Structural)
  - Tests derived from code structure - Code Based
  - Tests are evaluated in terms of coverage of the code structures

16

Black Box vs White Box Testing



17

Black vs White-Box: Which is Better?

→ **Spec: *evenORodd* returns 0 if number is even, -1 if it is odd.**

→ **Code:**

```
int evenORodd (int number) {
    int result;
    result= number mod 2;
    return (result);
}
```

→ ***evenORodd* works for even numbers, not for odd numbers**
- **BB detects fault (returns 1 for odd numbers instead of –1)**
- **WB with 100% coverage does not detect fault**

18

Black vs White-Box: Which is Better?

- Spec: *Division* prints the result of a division operation for integers
- Code:

```
Division (int numerator, int denonimator) {
    if (denonimator != 0)
        printf( "%d", numerator/ denonimator);
    else
        printf(" Failure ");
}
```

- *Division* only works for denominator !=0
  - BB may not detect the fault (all inputs are non-zero integers)
  - WB with 100% statement coverage would detect the fault

Coverage

- Make sure tests *cover* each part of program
  - Every statement
  - Every branch
  - Every path
  - Every condition
  - Every pass through a loop
- Measures the quality of tests
- How much of the program do the tests cover?

Statement Coverage Adequacy

```
PROGRAM GCD
begin
1  read(x)
2  read(y)
3  while (x != y) do
4        if (x > y) then
5            x = x - y
         else
6            y = y - x
         endif
   endwhile
7  print x
end
```

Statement coverage: find test cases that ensure coverage of every executable program statement

Try:

x = 8, y = 4
x = 2, y = 2
x = 2, y = 4

Statement Coverage – A Weakness

```
1  read(x)
2  read(y)
3   if (x > y) then
4       x = x - y
5   endif
6   print x
end
```

What's the weakness?

```
1. x = 0;

2. if  (input = 1)
       x = 2;

z = y/x;
```


Decision Coverage Adequacy (Review)

```
PROGRAM GCD
begin
1  read(x)
2  read(y)
3  while (x != y) do
4        if (x > y) then
5            x = x - y
         else
6            y = y - x
         endif
   endwhile
7  print x
end
```

Decision coverage: find test cases that ensure coverage of every outcome of each predicate (decision) in the program.

Try:

x = 8, y = 4
x = 2, y = 2
x = 2, y = 4

Decision Coverage – A Weakness (Review)

```
1  read(x)
2  read(y)
3   if (x > y && y > 0) then
4       x = x - y
5   Endif
6   z = A[y]
7   print z
end
```

What's the weakness?

Condition Coverage Adequacy

```
1  read(x)
2  read(y)
3   if (x > y && y > 0) then
4       x = x - y
5   endif
6   z = A[y]
7   print z
end
```

Condition coverage: find test cases that ensure coverage of every outcome of each predicate (decision) in the program, under each assignment of truth values to the individual conditions in that predicate.

Try:

x = 4, y = 2
x = 2, y = 4
x = -2, y = -4
x = -4, y = -2

Condition Coverage Issues

- How to instrument code to measure condition coverage?
- How many tests are needed for
  - (P || Q)
  - (P || Q || R)
  - (P || Q || R || S)

Some White Box Adequacy Criteria

- Statement coverage
- Decision coverage
- Condition coverage
- Path coverage

Summary

→ Testing ≠ debugging

→ Tests are code, too

→ A "quality" focus & practical concerns ("how many tests are needed?")

→ To-do

- Lab3 submission on GitHub: before 5pm, Tuesday (4/19)
- Lab 3 slides due: before 5pm, Wednesday (4/20)
- Be ready for your presentation on Thursday (4/21)

EECE3093C: Software Engineering

(Spring 2022)

Prof. Tingting Yu (tingting.yu@uc.edu)

Office hours via Zoom or in-person (806B Rhodes Hall):

email the instructor to make an appointment

1

Today's Menu

Last Week:
Lab3 Phase 1
Software Testing

This Week:
Lab 3 due
Software Laws
Summary

The End!

2

True or False

- If we get behind schedule, we can add more programmers and catch up.

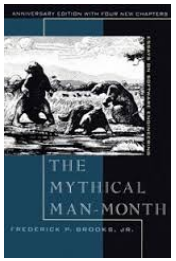
3

True or False

- If we get behind schedule, we can add more programmers and catch up.
- False
- Reality: Adding people to a late software project makes it later [Brooks' s law].

4

Fred Brooks



5

Brooks's Law

Brooks's Law: Adding manpower to a late software project makes it later.

– Frederick P. Brooks Jr.

6

Lab3: Testing Brooks's Law (Spring 21)

→ 9 days (3/30-4/9)

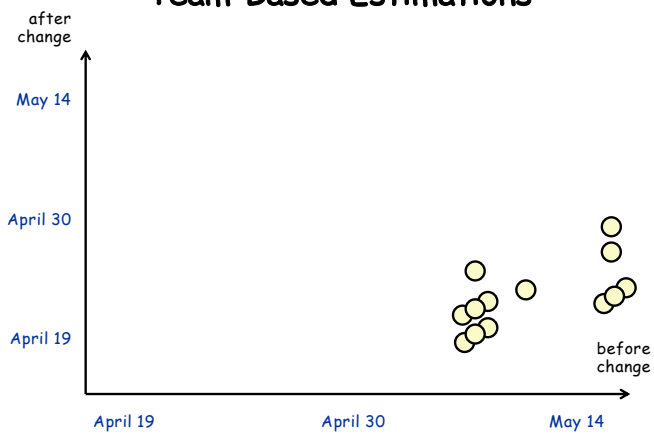
- 14 of 27 teams completed 33% or less
- Estimated completion day *(before change)*
- Made the members of the 13 teams join *different* teams
- Don't know anybody while "a late project team knows each other"
- Communication and coordination already established

→ 5 days (4/12-4/16)

- Estimated completion day *(after change)*

7

Team-Based Estimations



8

Testing Brooks's Law: Limitations

→ The new member(s) knew the answer(s)

9

From team-based to individual-based

→ 14 days (3/29-4/12) vs. 7 days (4/12-4/19)

↳ Completion percentage per day per person

before change	after change	before change	After change
0.83%	1.33%	1.83%	2.23%
1.22%	1.23%	1.22%	1.23%
1.22%	2.23%	1.22%	1.20%
0.93%	1.46%	0.69%	1.43%
1.22%	1.23%	0.93%	2.33%
1.83%	1.83%	0.93%	1.87%
0.92%	2.07%		

10

Lehman's Laws of Software Evolution

→ Conservation of Familiarity

↳ The amount of change in successive releases is roughly constant

before change	after change	before change	After change
0.83%	1.33%	1.83%	2.23%
1.22%	1.23%	1.22%	1.23%
1.22%	2.23%	1.22%	1.20%
0.93%	1.46%	0.69%	1.43%
1.22%	1.23%	0.93%	2.33%
1.83%	1.83%	0.93%	1.87%
0.92%	2.07%		

11

Lehman's Laws of Software Evolution

Source: Adapted from Lehman 1980, pp1061-1063

→ Continuing Change

↳ Any software that *reflects some external reality* undergoes continual change or becomes progressively less useful

➢ change continues until it is judged more cost effective to replace the system

→ Increasing Complexity

↳ As software evolves, its *complexity* increases…

➢ …unless steps are taken to control it.

→ Conservation of Familiarity

↳ The amount of change in successive releases is roughly constant

12

What makes software so SPECIAL?

- → Abstract, invisible, intangible
  - Its purpose is to configure some hardware to do something useful
  - Software is used and experienced
- → Never wears out
  - Defects are NOT arising after software has been used often
  - Software follows its own laws other than physical laws
- → Is changing all the time
  - Whenever there is a bug, there is a patch
  - Software maintenance costs a lot more than initial builds

"Software is not limited by physics, like building are. It is limited by imagination, by design, by organization. In short, it is limited by properties of people, not by properties of the world. *We have met the enemy, and he is us.*"

Ralph Johnson

13

Last Lecture

- → Presentations
- → Final exam

14

1. Sign in or Sign up at https://github.com/
2. Create a repository on GitHub:

About repositories:

A repository contains all of your project's files and each file's revision history. You can discuss and manage your project's work within the repository.

- Click icon in the upper right corner, then choose “your repositories”. (Fig 1)
- Click “New” button to create new repositories: (Fig 2)

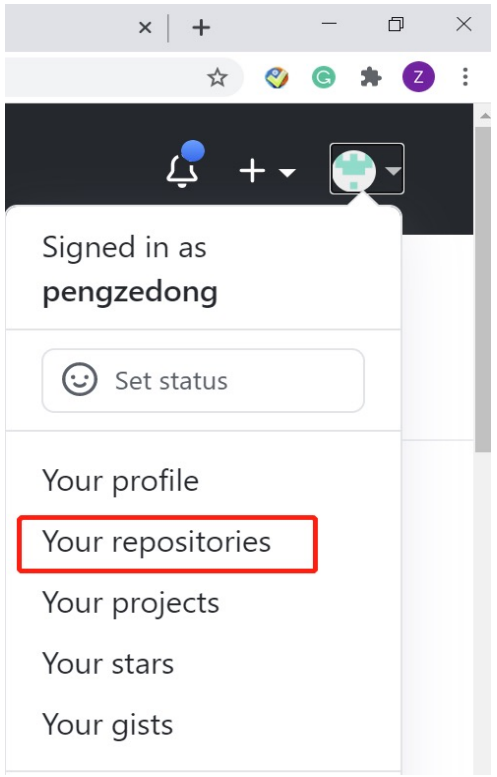


Fig 1

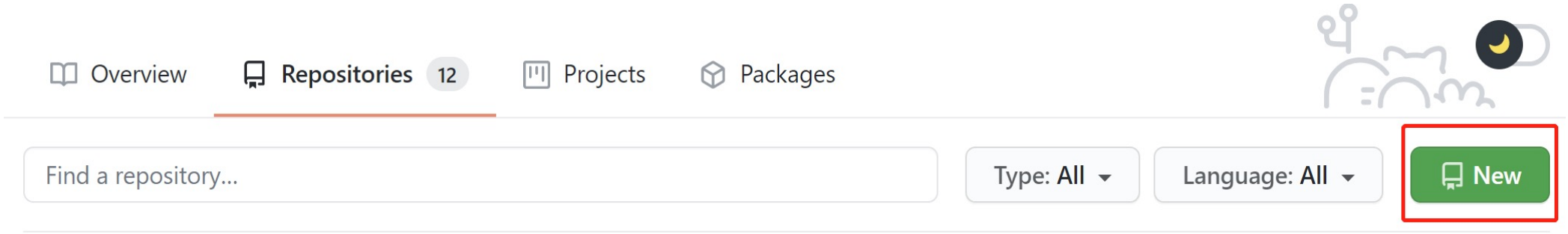
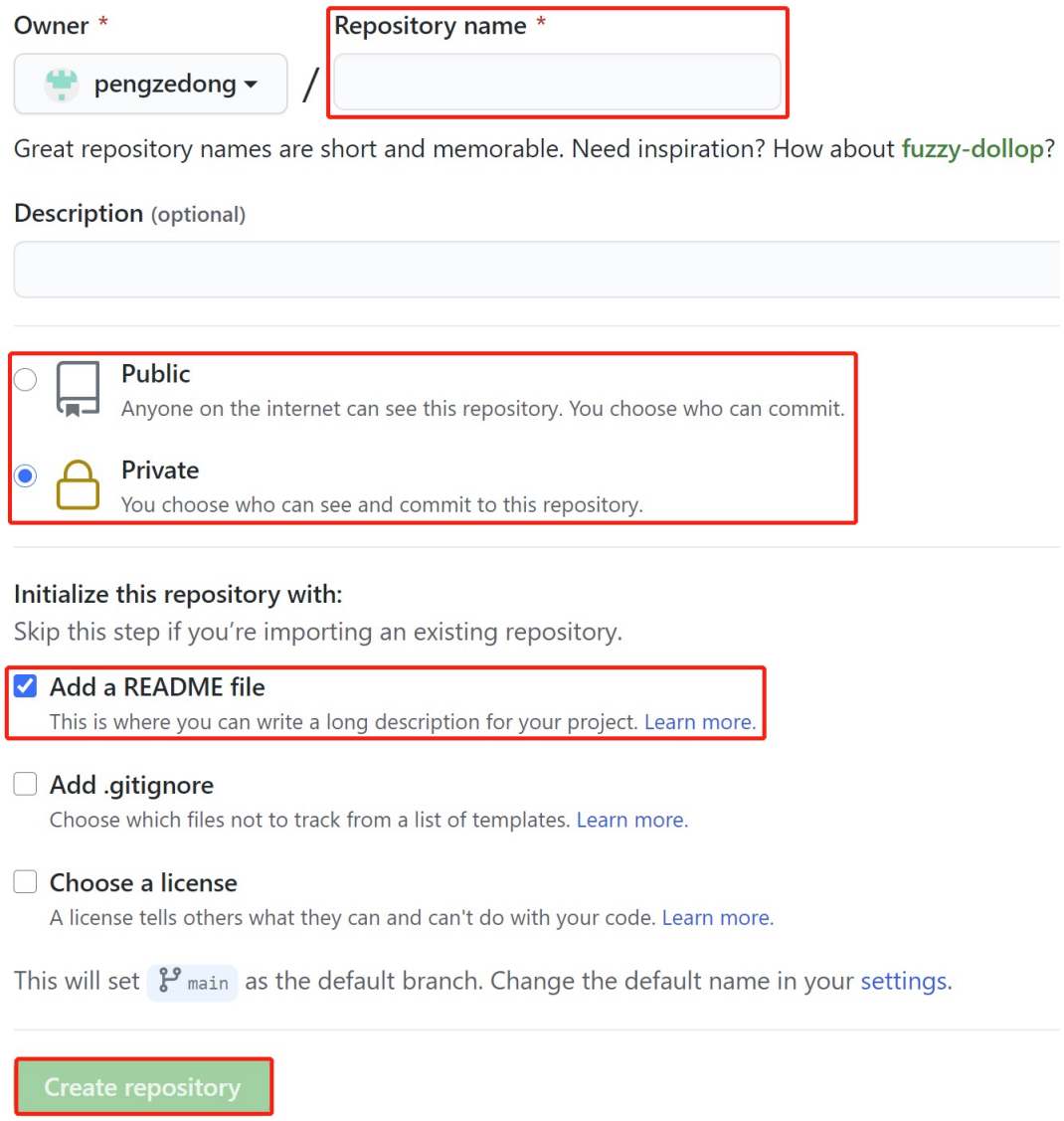


Fig 2

- Repository name, e.g., “EECE3093C-001-Group1”
- Please make sure that your repository is “Private”, so that other groups will not see your work. Inviting group members is introduced in the next slide.
- Add a README file for writing some descriptions about repository.
- Click “Create repository” to create repository.

- Note that for *Quiz3’s first question*, please name your repository to be: **“EECE3093C-SP22-Lab1”**



Private repository invite

- ❑Click “Setting” → Click “Manage access”
- ❑Click “invite a collaborator” to send invite link (Fig1)
- ❑In Fig 2, typing one of collaborator’s username, full name, or register email to send invite (**NOTE**: the collaborator also needs to have a GitHub account)
- ❑Please make sure to invite the instructor **(tingting703)** and the TAs (**pengzedong, mdarban66)** to your repository.

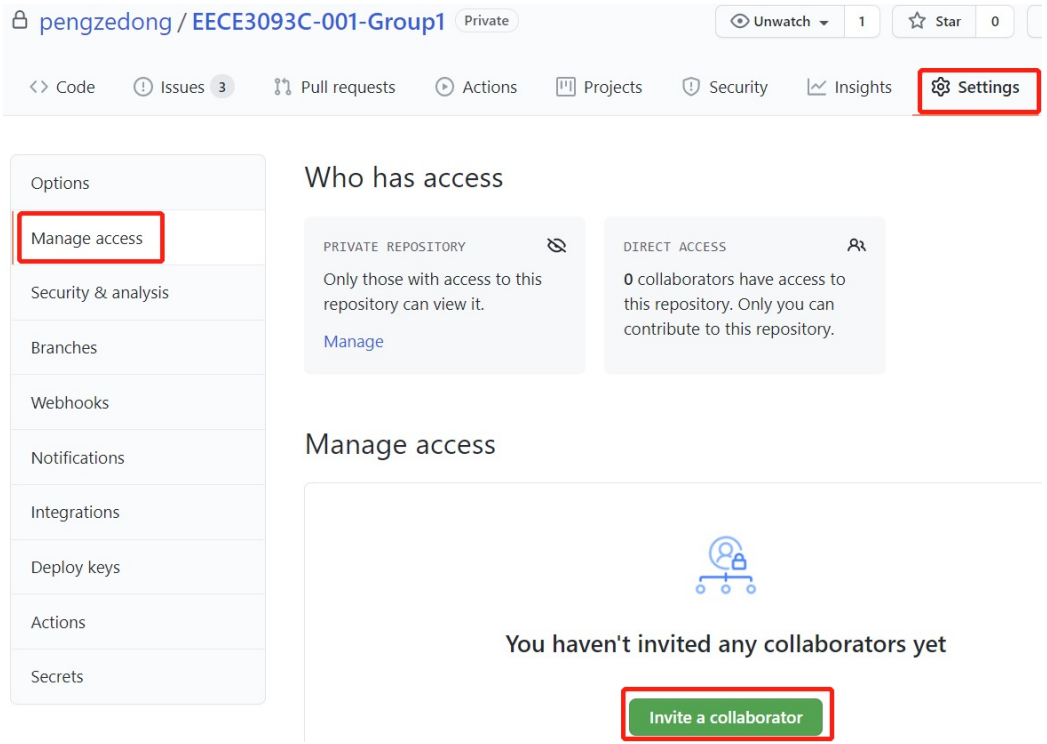


Fig 1

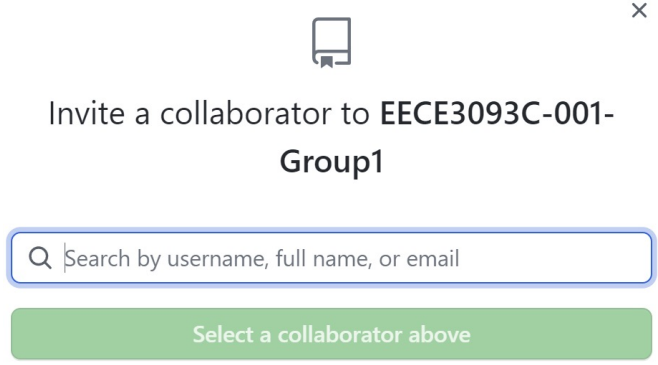


Fig 2

Upload files:

❑Click “ < > Code ” → “ Add file”

→“Upload files” (Fig 1)

❑Drag or choose files at upload page → click “Commit changes” to upload files or folder (Fig 2). (Add description for each upload will help for teamwork but not requited for upload)

Edit README file:

❑Click ✎ to Edit README file for addition information (Fig 1), e.g., project detail, group members name

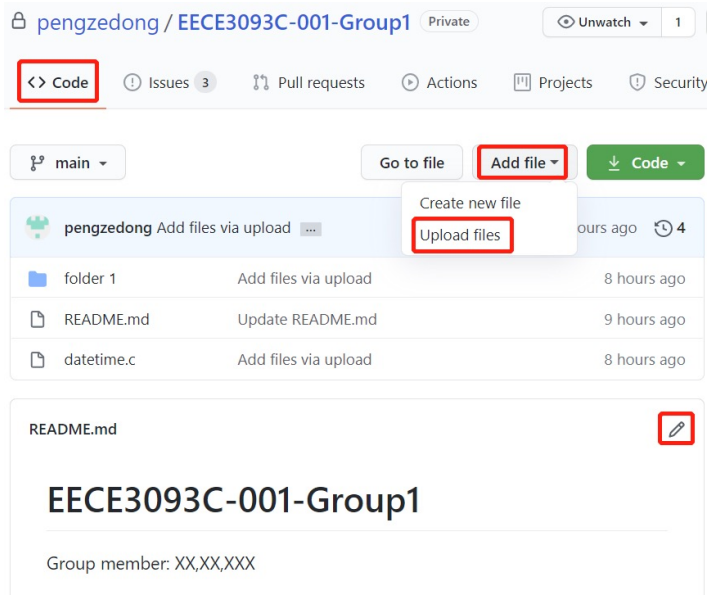


Fig 1

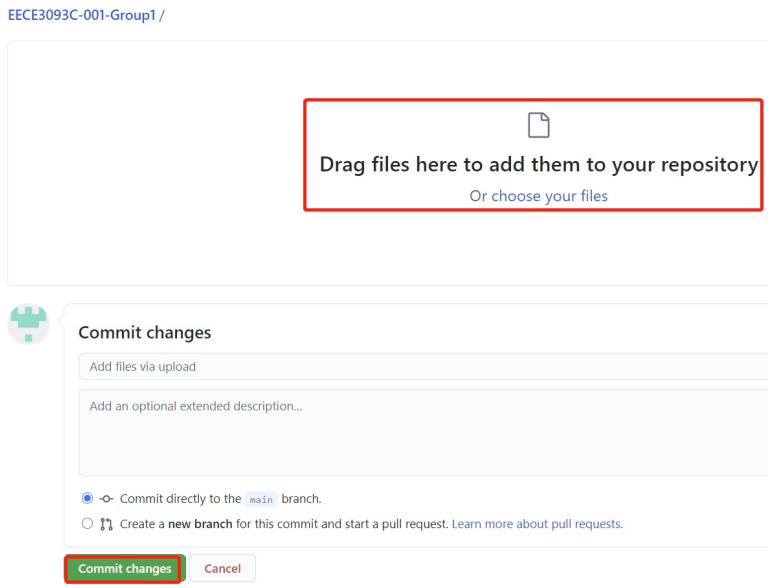


Fig 2

Edit file

❑Click “ ” to edit file

Delete file or folder

❑ Open the folder (Folder1) in GitHub then go to list of files and then delete one by one by clicking on delete icon highlighted in below Fig. **When last file in folder delete then the folder (Folder1) will be deleted automatically.**

About issue: issues are suggested improvements, tasks or questions related to the repository.

Add issues

- ❑ Click “Issues” → click “New issue” to create issue (Fig 1)

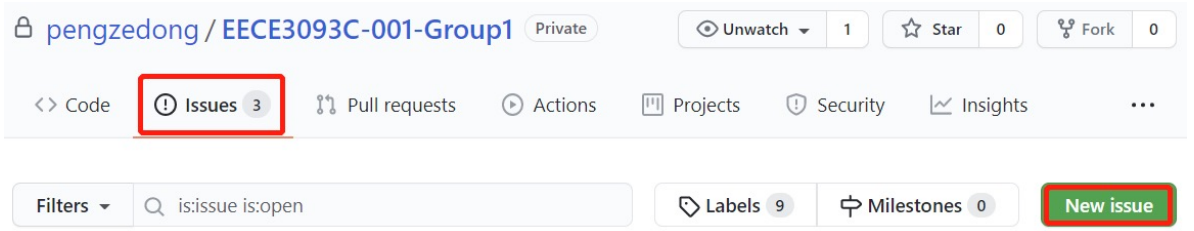


Fig 1

- ❑ Typing title and comment, add issues by click “Submit new issue” (Fig 2)

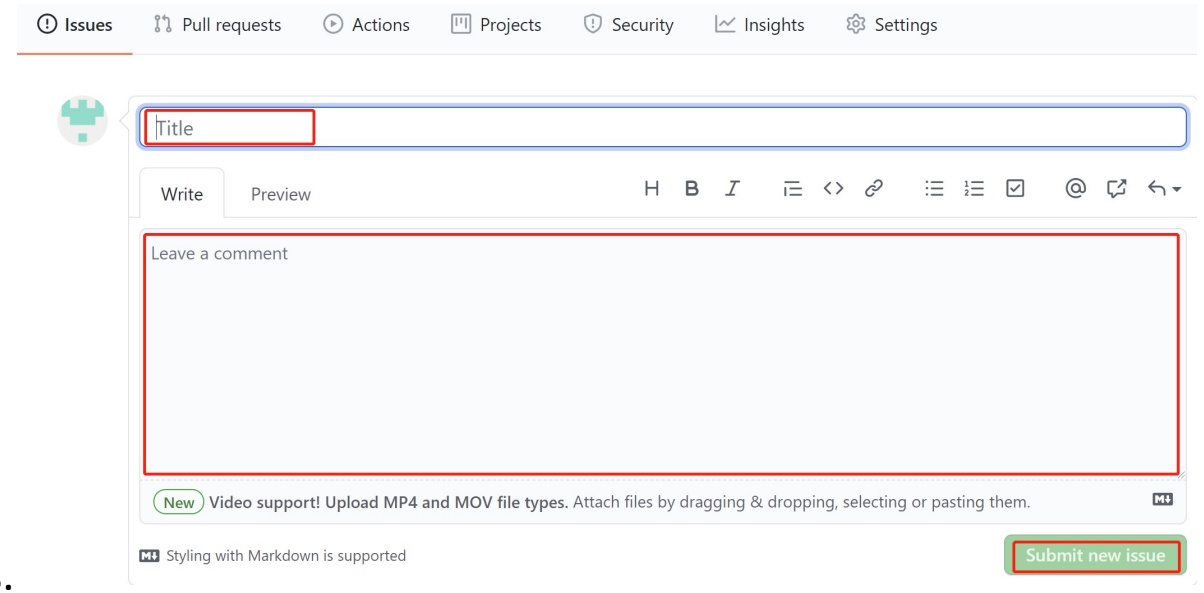


Fig 2

- ❑ It is of vital importance that ***ALL*** the members of ***ALL*** the groups are using the issues (e.g., creating, assigning, comments, closing, re-opening, etc.) to record group-wide communications toward the completion of each lab. The issues are integral to the ***grading*** of labs.

Add comment in issue

- ❑ Leave comment and click “comment” to add comment

Close and reopen issue

- ❑ Issue can close and reopen by click “Close issue” and “Reopen issue”

Delete issue

- ❑ Click “Delete issue” to delete issue permanent (cannot be undone; only administrators can delete issues)