CSCE 240: Advanced Programming Techniques

Lecture 13: Review Object Oriented Concepts – Inheritance, Polymorphism

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

20TH FEBRUARY 2024

Carolinian Creed: "I will practice personal and academic integrity."

Credits: Some material reused with permission of Dr. Jeremy Lewis. Others used as cited with thanks.

Organization of Lecture 13

- Introduction Section
  - Recap of Lecture 12
- Main Section
  - Review: Inheritance
  - Review: Polymorphism
  - Review: Regex
  - Review: Constructors and Destructors
- Concluding Section
  - About next lecture – Lecture 14
  - Ask me anything

Introduction Section

Recap of Lecture 12

- We looked at the concepts of
  - constructor
  - Destructor
- PA #2 was due

AAAI 2024 and AI Research on Campus

- Overall: https://aaai.org/aaai-conference/
  - Under-Graduates program: https://aaai.org/aaai-conference/undergraduate-consortium-program/
- AI4Society at AAAI 2024 (https://ai4society.github.io/) :
  - Tutorial (LLMs for planning), a workshop (AI and elections), a deployed application paper (ULTRA), and a demonstration paper (AI planning for information spread in social networks).
  - Looking for 1-2 undergrads to work in summer on AI algorithmic issues; send note/ talk to instructor; funding - on-campus funding programs (McNair/ Magellan/ …) + top-up by instructor

Main Section

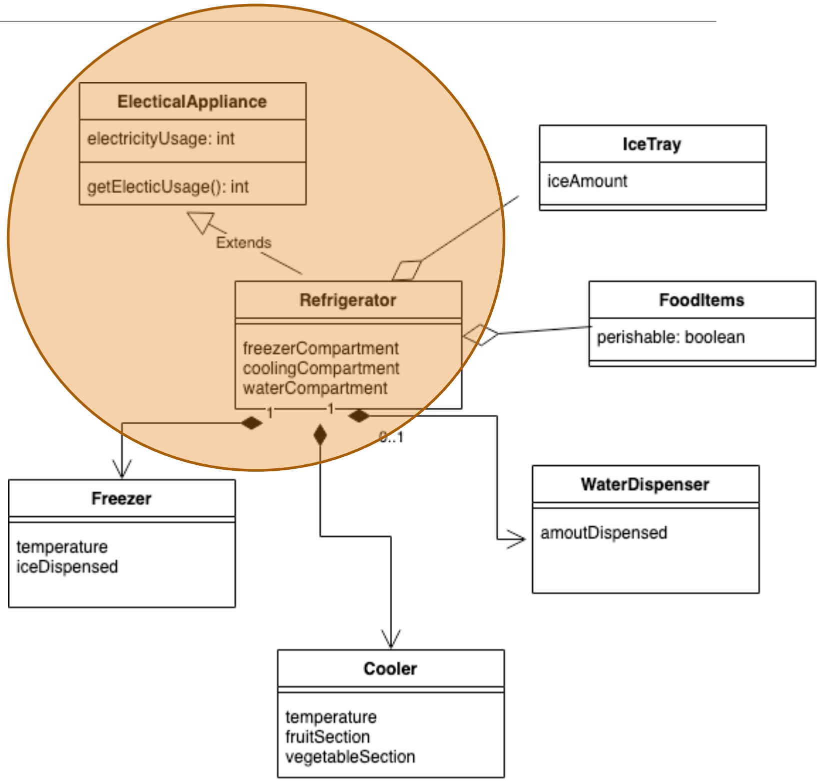
Review of Concept: Inheritance

What is Inheritance ?

- A class “inheriting” or reusing **characteristics** from another, existing class
- Synonyms: subclassing, specialization, derived
- Analogy: child inheriting from a parent
  - “Course-CSCE-240” sub-class of “Course-Undergraduate”
  - “USA” specialization of “Country”
- What are characteristics
  - Data members
    - Enrollment, timing, syllabus: course domain
    - Capital, head-of-state, currency: country domain
  - Functions manipulating the data members

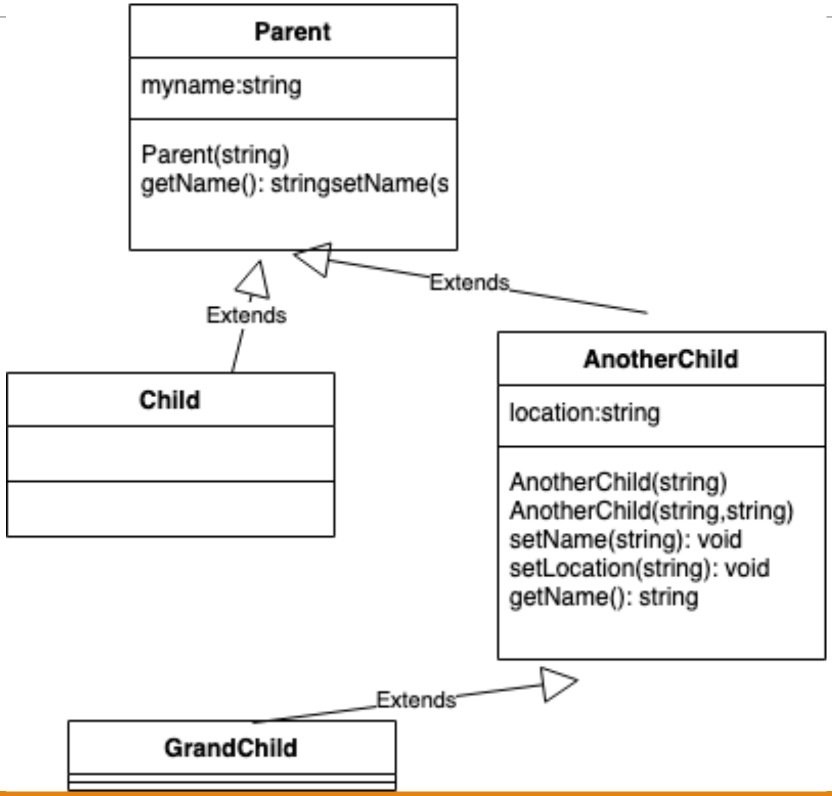
Why Use Inheritance ?

- Promote reuse
- Make code understandable, improve maintainability
- Promote security and data integrity
- Improve testing
- Improve code development productivity



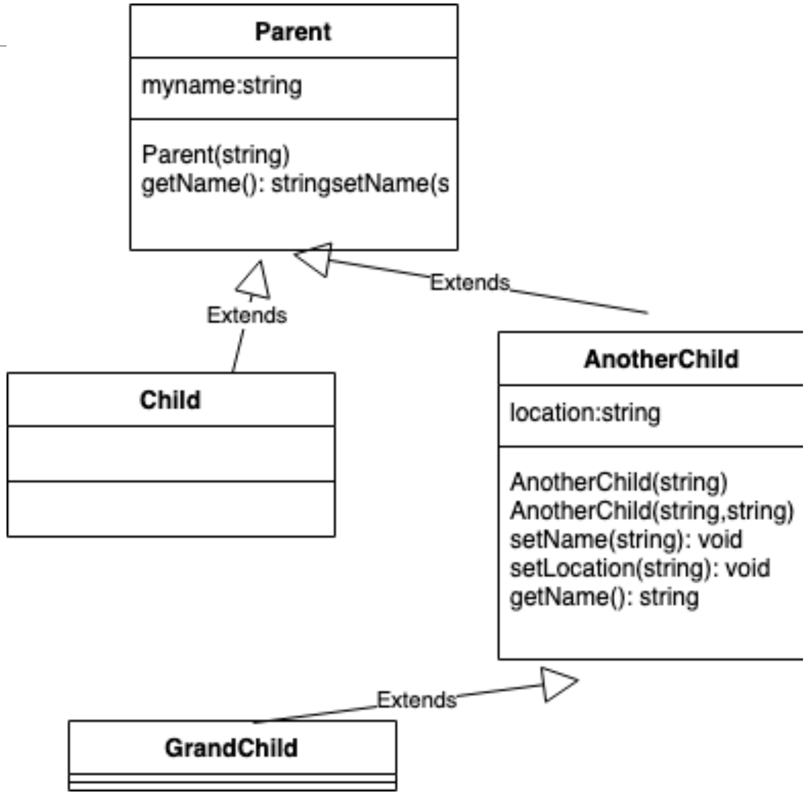
How to Use Inheritance ?

- Language independent syntax
- Illustration
  - 4 classes
  - 2 data members: myname, location
  - Access restrictions: private, protected, public



Notes on Inheritance

- Code for classes Child and GrandChild are minimal
  - Code reuse happens by default
- A child can override the behavior of its parent



Inheritance Type

- The access control levels (public, protected and private) in a class can be modified by inheritance types.
- Three inheritance types: public, protected, private
  - In public, all methods and members inherited from the parent maintain their access control level
  - In protected, all methods and members inherited from the parent maintain protected or lower access control level
  - In private, all methods and members inherited from the parent maintain private access control level
- **By default, we had been working with public inheritance types**

Access \ Inheritance Type	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	private	private	private

Running Example - Name

- Scope: class diagram to cover names for humans and robots
  - Members: First, Middle, Last, Prefix, Suffix, Nickname, PreferredName
  - Methods: Getters and Setters
- Classes: AllNames, HumanNames, RobotNames
- Possible design
  - AllName: (PreferredName)
  - HumanNames (First, Middle, Last, Prefix, Suffix, Nickname)
  - RobotNames (PreferredName)
- Exercise:
  - Think of access and inheritance types

Review of Concept: Polymorphism

“Multiple shapes”

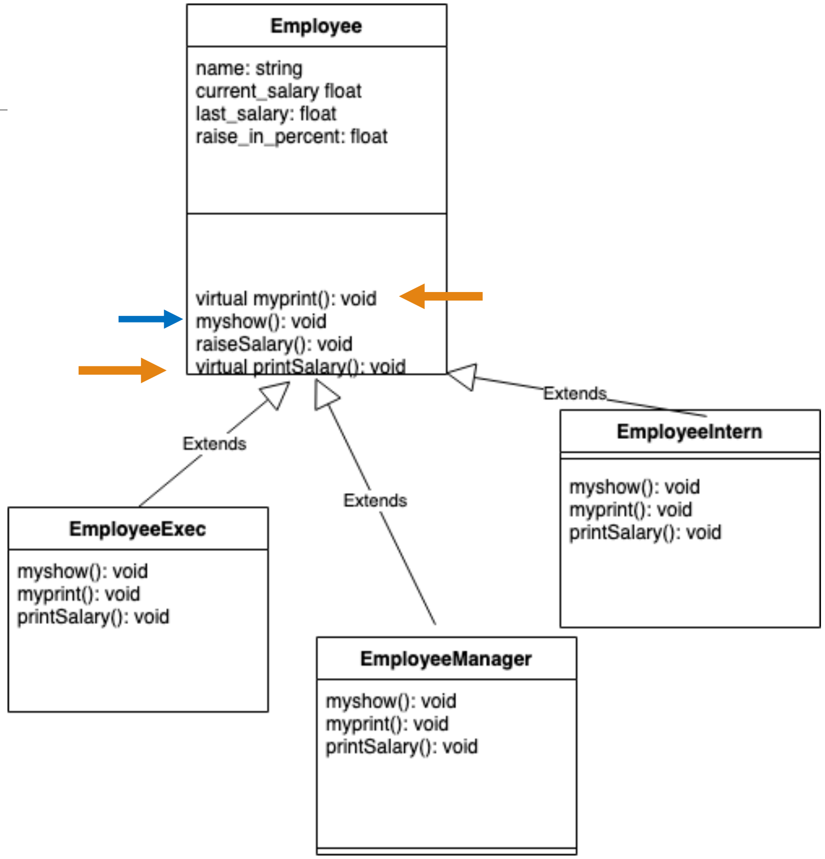
What is Polymorphism ?

- A class “inheriting” or reusing **characteristics** from another, existing class, dynamically depending on how the method is declared !
- In contrast, inheritance discussed until now was static

Why Use Polymorphism ?

- Promote reuse
- Make code understandable, improve maintainability
- Promote security and data integrity
- Improve testing
- Improve code development productivity
- **Context-dependent customization of inheritance**

Code: https://github.com/biplav-s/course-adv-proglang/tree/main/sample-code/CandC%2B%2B/Class9and10_C%2B%2B_OOAdv/src



Employee

name: string
current_salary float
last_salary: float
raise_in_percent: float

virtual myprint(): void
myshow(): void
raiseSalary(): void
virtual printSalary(): void

EmployeeExec

myshow(): void
myprint(): void
printSalary(): void

EmployeeManager

myshow(): void
myprint(): void
printSalary(): void

EmployeeIntern

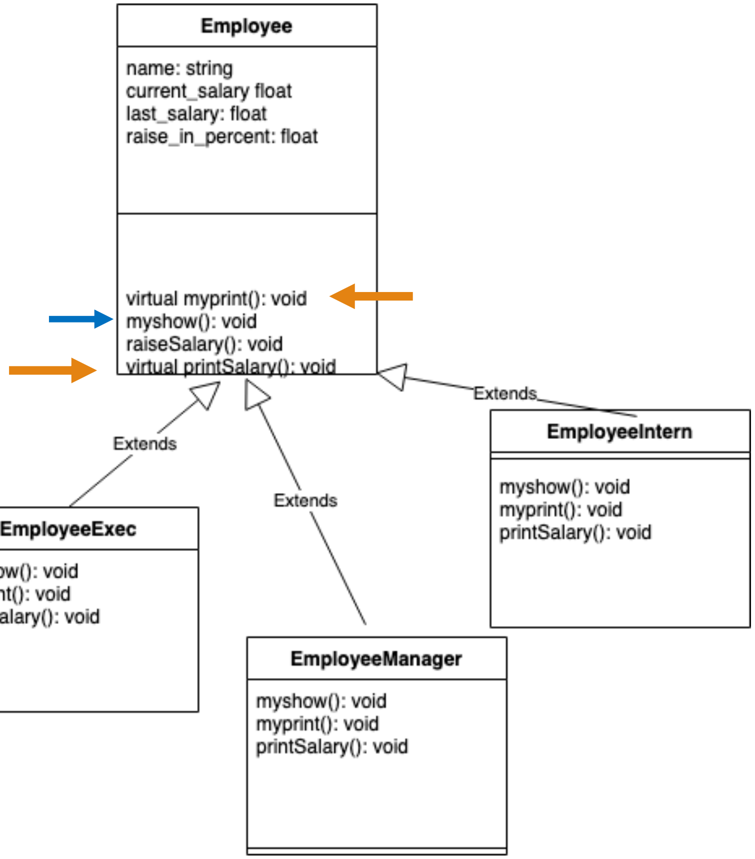
myshow(): void
myprint(): void
printSalary(): void

```
Credits: Based on code at
- https://www.geeksforgeeks.org/polymorphism-in-c/
- https://www.geeksforgeeks.org/virtual-functions-and-runtime-polymorphism-in-c-set-1-introduction/
```

How to Use Polymorphism ?

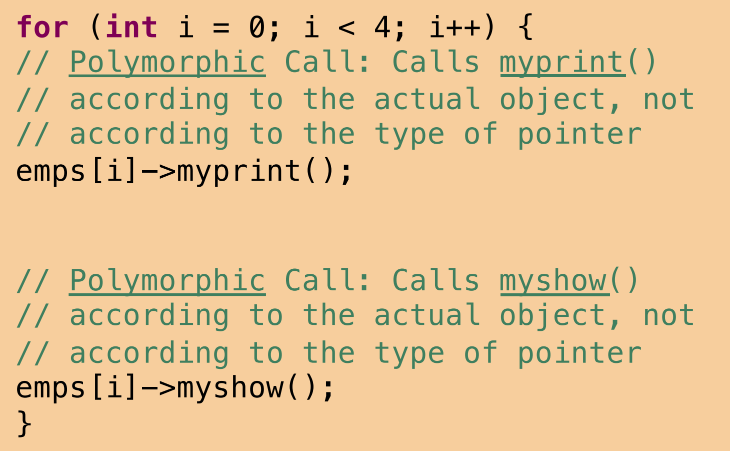
- Language independent syntax
- Illustration
  - 4 classes; 1 base, 3 derived
  - Basic: no data members; myshow() and myprint() functions
  - Advanced: 3 data members, printSalary() function

```
Employee: myprint base class
Employee: myshow base class

EmployeeManager: myprint derived class
Employee: myshow base class

EmployeeIntern: myprint derived class
Employee: myshow base class

EmployeeExec: myprint derived class
Employee: myshow base class
```

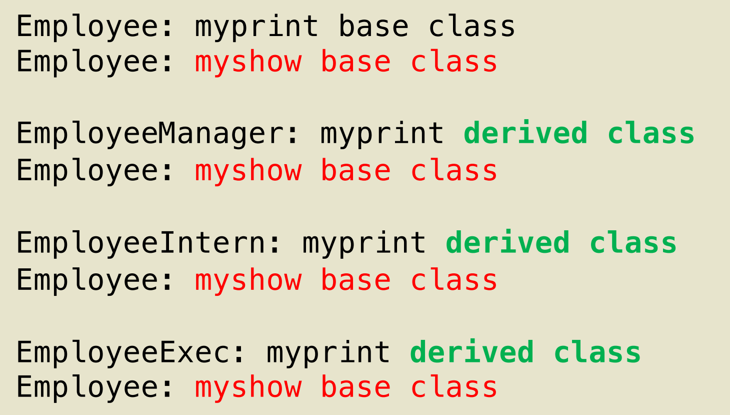


Key Points - Polymorphism

1. The method must appear in a class that is part of an inheritance hierarchy
2. The method must declared virtual in the base class at the top of the hierarchy
3. Derived classes override the behavior of the inherited virtual methods as needed.
4. Clients must invoke the method via a pointer (or reference) to an object, not directly through the object itself



```
for (int i = 0; i < 4; i++) {
// Polymorphic Call: Calls myprint()
// according to the actual object, not
// according to the type of pointer
emps[i]->myprint();


// Polymorphic Call: Calls myshow()
// according to the actual object, not
// according to the type of pointer
emps[i]->myshow();
}
```



```
Employee: myprint base class
Employee: myshow base class

EmployeeManager: myprint derived class
Employee: myshow base class

EmployeeIntern: myprint derived class
Employee: myshow base class

EmployeeExec: myprint derived class
Employee: myshow base class
```

Code: https://github.com/biplav-s/course-adv-proglang/tree/main/sample-code/CandC%2B%2B/Class9and10_C%2B%2B_OOAdv/src

Credit: Fundamentals of Programming C++, Richard L. Halterman

Notes on Polymorphism

- Support for Polymorphism is not uniform across languages
- C++ is most expressive; controlled by virtual; allows dynamic binding (change of behavior)
- Java and Python have limited support; does static binding

```
Employee: myprint base class
Employee: myshow base class

EmployeeManager: myprint derived class
Employee: myshow base class

EmployeeIntern: myprint derived class
Employee: myshow base class

EmployeeExec: myprint derived class
Employee: myshow base class
```

Review of Concept: Regex

“Regular Expressions”

Review:
Regular Expression

<table>
<tr><th>Metacharacter</th><th>Explanation</th></tr>
<tr><td>^</td><td>Matches the starting position within the string</td></tr>
<tr><td>.</td><td>Matches any single character</td></tr>
<tr><td>[ ]</td><td>Matches a single character that is contained within the brackets</td></tr>
<tr><td>[^]</td><td>Matches a single character that is not contained within the brackets.</td></tr>
<tr><td>$</td><td>Matches the ending position of the string</td></tr>
<tr><td>*</td><td>Matches the preceding element zero or more times</td></tr>
<tr><td>+</td><td>Matches the preceding element one or more times</td></tr>
<tr><td>|</td><td>Separates choices</td></tr>
</table>

<table>
<tr><th>Regex</th><th>Matches any string that</th></tr>
<tr><td>hello</td><td>contains {hello}</td></tr>
<tr><td>gray|grey</td><td>contains {gray, grey}</td></tr>
<tr><td>gr(a|e)y</td><td>contains {gray, grey}</td></tr>
<tr><td>gr[ae]y</td><td>contains {gray, grey}</td></tr>
<tr><td>b[aeiou]bble</td><td>contains {babble, bebble, bibble, bobble, bubble}</td></tr>
<tr><td>[b-chm-pP]at|ot</td><td>contains {bat, cat, hat, mat, nat, oat, pat, Pat, ot}</td></tr>
<tr><td>colou?r</td><td>contains {color, colour}</td></tr>
<tr><td>rege(x(es)?|xps?)</td><td>contains {regex, regexes, regexp, regexps}</td></tr>
<tr><td>go*gle</td><td>contains {ggle, gogle, google, gooogle, goooogle, ...}</td></tr>
<tr><td>go+gle</td><td>contains {gogle, google, gooogle, goooogle, ...}</td></tr>
<tr><td>g(oog)+le</td><td>contains {google, googoogle, googoogoogle, googoogoogoogle, ...}</td></tr>
<tr><td>z{3}</td><td>contains {zzz}</td></tr>
<tr><td>z{3,6}</td><td>contains {zzz, zzzz, zzzzz, zzzzzz}</td></tr>
<tr><td>z{3,}</td><td>contains {zzz, zzzz, zzzzz, ...}</td></tr>
</table>

Example Source: https://cs.lmu.edu/~ray/notes/regex/

Regex - Code Demo

Code: https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class9and10_C%2B%2B_OOAdv/src/Class9and10_C%2B%2B_OOAdv.cpp

Argument: [^0-3]

What does this mean?

Review of Concepts: Constructors and Destructors

Constructor - What is It?

- Special function in every class
  - Always has the same name as the class itself
  - Does not have an explicit return type
  - Multiple constructors possible per class

- **Purpose**: Used to initialize objects of that class

Specification

```
class PersonName {
            string firstName;
            string lastName;

public:
            PersonName();
            PersonName(string);
            PersonName(string, string);
…
```

https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class7and8_C%2B%2B_OO/src/headers/PersonName.h

Constructor - What is It?

- Special function in every class
  - Always has the same name as the class itself
  - Does not have an explicit return type
    Multiple constructors possible per class

- **Purpose**: Used to initialize objects of that class

Usage

```
PersonName p1;
PersonName p2("Joginder");
PersonName p3("Joginder", "Singh"
```

Implementation

```
PersonName::PersonName() {
            firstName = "default-Maria";
            lastName = "default-Wang";
}

PersonName::PersonName(string first) {
            firstName = first;
            lastName = "default-Wang";
}

PersonName::PersonName(string first,
                         string last) {
            firstName = first;
            lastName = last;
}
```

https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class7and8_C%2B%2B_OO/src/implem/PersonName.cpp

Destructors - What is It?

- Special function in every class
  - Always has the same name as the class itself but prefixed with ~
  - Does not have an explicit return type
  - Does not take an argument
  - Maximum one destructor per class

- **Purpose**: Used to cleanup before removing objects of that class
  - Common usage: freeing memory allocated by the object's data members before the object is destroyed
  - Common usage: Close files, streams

Implementation

```
PersonName::~PersonName() {

}
```

https://github.com/biplav-s/course-adv-proglang/blob/main/sample-code/CandC%2B%2B/Class7and8_C%2B%2B_OO/src/implem/PersonName.cpp

Discussion: Using Constructors / Destructors Effectively

- Remember: Create automatically if none provided by developer
- Constructor: initialization of data members
- Destructor: clean-up
- Remember the order, use it productively but do not overly depend on it.

Discussion: Course Project

Course Project – Knowing About Companies

- **Project**: Develop collaborative assistants (chatbots) that offer useful information about companies
- Specifically, use the EDGAR dataset on companies at: https://www.sec.gov/edgar/searchedgar/companysearch.
    - For Apple, it is: https://www.sec.gov/edgar/browse/?CIK=320193&owner=exclude
- **Each student will choose two companies (from thousand available).**
- Programming assignment programs will: (1) extract data about two companies from 10-k, (2) process it, (3) make content available in a command-line interface, (4) handle any user query and (5) report on interaction statistics.

Core Programs Needed for Project

- Prog 1: extract data from the district **[prog1-extractor]**
- Prog 2: process it (extracted data) based on questions **[prog2processor]**
- **Prog 3: make content available in a command-line interface [prog3-ui]**
- Prog 4: handle any user query and
- Prog 5: report statistics on interaction of a session, across session

Programming Assignment # 3

- **Goal**: **make content available in a command-line interface** [Name: **prog3-ui**]
- Program should do the following:
  - Run in an infinite loop until the user wants to quit
  - Handle any user response
    - User can quit by typing "Quit" or "quit" or just "q"
    - User can enter any other text and the program has to handle it. The program should write back what the user entered and append, saying – **" – I do not know this information**".
  - Handle known user query types
    - "Tell me about ***IBM***" or "What are the risk factor for ***IBM***?" => (Part 1), or (Part 1: Item 2), accordingly
    - "What markets does ***IBM*** operate in?", "Are there aby disclosures from ***IBM***?" => (Part 2)
    - "who are the directors? " => (Part 3: Item ..) // assume company, or tell of all companies, or ask …
    - "Tell me about ***IBM's*** statements" => (Part 4)
    - …
    - "**Tell me everything**" => ***Give all information extracted***

Concepts: 10-K, Parts, Items

Parts

- Part 1: Business Background and Risks
  - Item 1: Business
  - Item 2: Risk factors
  - Item 3: Properties
  - Item 4: Legal Proceedings
- Part 2: Operations and Disclosures
  - .. Market
  - .. Disclosures
- Part 3: Company Structure
  - Directors
  - Compensation
- Part 4: Financial Statements
  - Statements

Notes on PA#3

- Handle all parts and items
  - Multiple ways to ask for same information type
  - Variant assumes company’s name from context or is specified
- Handle special query: ***Tell me everything***
- Handle others
  - Chit-chat
  - Give controlled response under all condition

Programming Assignment # 3

- Code organization
  - Create a folder in your GitHub called “prog3-ui”
  - Have sub-folders: src (or code), data, doc, test
  - Write a 1-page report in ./doc sub-folder
  - Send a confirmation that code is done by updating Google sheet; optionally, send email to instructor and TA

- Use concepts learned in class
  - Classes
  - Exceptions
  - UML Diagrams

Content Reference: Queries for (Answers) Data We Have

- What does the (company) do? // Answers in Part 1
  - What is the (company's) business?
  - What are (company's) risk factors?
  - What does (company) own?
  - …
- Where does (company) operate? // Answers in Part 2
  - What has (company) disclosed?
- How is (company) structured? // Answers in Part 3
  - Who is (company's) CEO?
  - How much does (person) earn?
  - …
- What was in (company) statements? // Answers in Part 4
  - …

Concepts: 10-K, Parts, Items

Parts

- Part 1: Business Background and Risks
  - Item 1: Business
  - Item 2: Risk factors
  - Item 3: Properties
  - Item 4: Legal Proceedings
- Part 2: Operations and Disclosures
  - .. Market
  - .. Disclosures
- Part 3: Company Structure
  - Directors
  - Compensation
- Part 4: Financial Statements
  - Statements

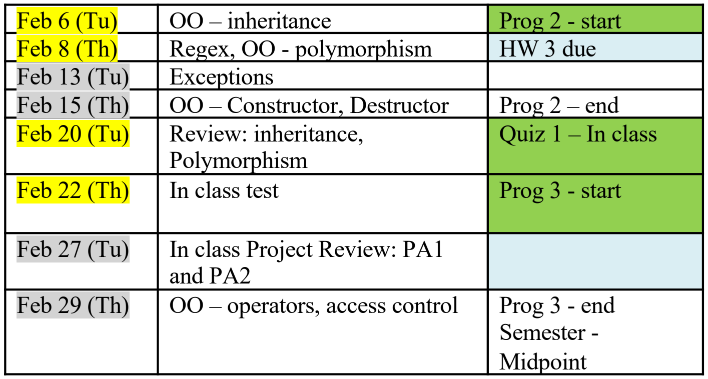
Concluding Section

Lecture 13: Concluding Comments

- Looked again at the concept of inheritance; covered inheritance type
- Looked again at the concept of polymorphism
- Looked at regex
- Looked at constructors and destructors
- Start of PA3

About Next Lecture – Lecture 14

Lecture 14: Quiz1



Feb 6 (Tu)	OO – inheritance	Prog 2 - start
Feb 8 (Th)	Regex, OO - polymorphism	HW 3 due
Feb 13 (Tu)	Exceptions	
Feb 15 (Th)	OO – Constructor, Destructor	Prog 2 – end
Feb 20 (Tu)	Review: inheritance, Polymorphism	Quiz 1 – In class
Feb 22 (Th)	In class test	Prog 3 - start
Feb 27 (Tu)	In class Project Review: PA1 and PA2	
Feb 29 (Th)	OO – operators, access control	Prog 3 - end Semester - Midpoint