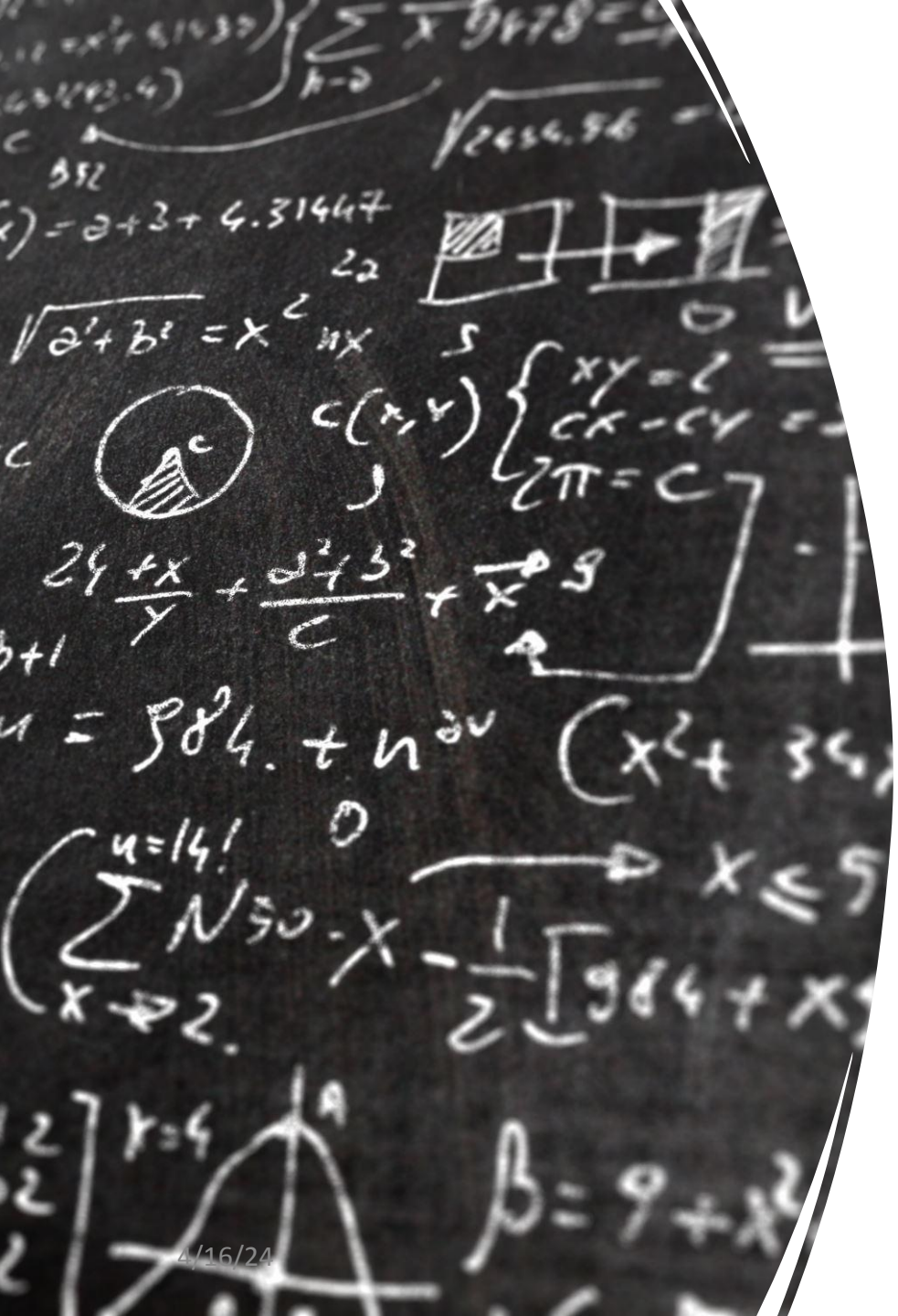

CS 113 – Computer Science I

Lecture 22 – Midterm 2 Review

Tuesday 4/16/2024

Announcements

Midterm next week (Tuesday April 23rd)

No lab on Thursday - extra credit opportunity instead

No OH next two Fridays (4/19 and 4/26)

Midterm 2

10 points multiple choice

10 points short answer

60 points programming

15 points reading code

80min

1 page cheat sheet front and back

Midterm 2 Topics

1. Arrays of Arrays
2. Nested Loops
3. Truth tables
4. Mutability
5. Classes
    a. constructors, accessors, modifiers, instance variables, this keyword
6. equals
7. toString
8. Statics vs non-static
9. OOP
    a. super keyword
    b. Access modifiers
    c. Inheritance and polymorphism
10. try-catch
11. interfaces

Arrays of Arrays

int[] array1 is an array of ints

String[] array2 is an array of Strings

What is int[][] array3?

An array of integer arrays

What is String[][] array4?

An array of String arrays

2D array example

What does int[][] array = new int[4][3] look like?

2D array example

What does int[][] array = new int[4][3] look like?

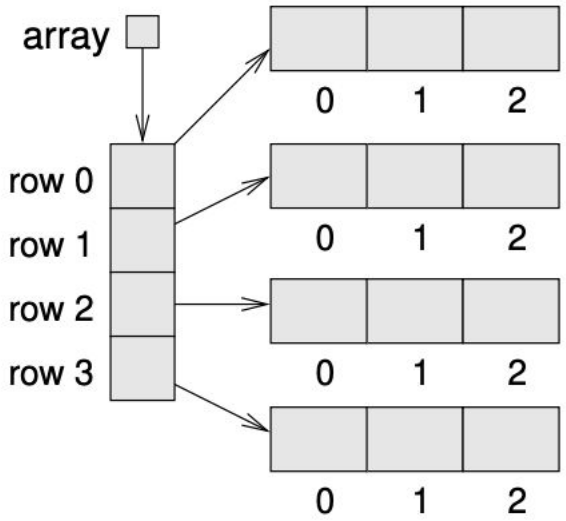

Figure 15.3: Storing rows and columns with a 2D array.

Arrays of Arrays

a) Write a line of Java code to access and print the element in the second row and third column of the matrix.

b) Write a line of Java code to calculate and print the sum of all elements in the matrix.

c) Write a line of Java code to check if the element at the second row and third column is equal to 6. Your code should print "true" if it is and "false" otherwise.

Arrays of Arrays

Write a program to check if all chars in a matrix of chars are vowels

Arrays of Arrays

You are tasked with creating a Java program to manage a simple locker system. The lockers are arranged in rows and columns, and each locker can be either empty or occupied.

You plan to implement the following functionalities:

1. **Initialization:** Write a method to initialize the locker system with a given number of rows and columns, filling it with default values indicating empty lockers (e.g., 'O' for empty and 'X' for occupied).
2. **Locker Occupancy:** Write a method to change the occupancy status of a specific locker (i.e., mark it as occupied or empty). If the locker is already in the desired state, print an appropriate message.
3. **Check Locker Status:** Write a method to check the occupancy status of a specific locker.
4. **Display:** Write a method to display the current state of the locker system, showing which lockers are occupied and which are empty.

Truth Tables

```
boolean yes = true;
boolean no = false;
int loVal = -999;
int hiVal = 999;
double grade = 87.5;
double amount = 50.0;
String hello = "world";
```

Expression	Result
yes == no || grade >amount	
amount == 40.0 || 50.0	
hiVal != loVal || loVal <0	
True || hello.length() >0	
hello.isEmpty() && yes	
grade <= 100 && !false	
!yes || no	
grade >75 >amount	
amount <= hiVal && amount >= loVal	
no && !no || yes && !yes	

```
boolean yes = true;
boolean no = false;
int loVal = -999;
int hiVal = 999;
double grade = 87.5;
double amount = 50.0;
String hello = "world";
```

Expression	Result
yes == no || grade >amount	T
amount == 40.0 || 50.0	err
hiVal != loVal || loVal <0	T
True || hello.length() >0	T or err
hello.isEmpty() && yes	F
grade <= 100 && !false	T
!yes || no	F
grade >75 >amount	err
amount <= hiVal && amount >= loVal	T
no && !no || yes && !yes	F

Truth Tables

TruthTables.java

Mutability

Mutable vs Immutable

Mutable:

- Values can change
- methods can change the state of the object directly

Immutable:

- Values cannot change
- Instead, any operations that would alter the object's state return a new object with the modified state

Strings and Integers are immutable

Mutable vs Immutable

claim: **Strings and Integers are immutable**

code!

The underlying value can change, but this will create a **new object**

Mutable vs Immutable Classes

Are the following mutable or immutable?

Student.java

StudentCourse.java

Instructor.java

Classes

Classes

1. **constructors**
2. instance variables
3. accessors
4. modifiers
5. this keyword

Constructors

- Special method with same name as the class

- Initializes the newly created object

Creating objects

How do we create a new object of our class type?

Create by calling constructor using `new`

```
Scanner sc = new Scanner(System.in);

Instructor i = new Instructor(“elizabeth”, “Park”, 205)
```

Constructors

What is a *default constructor*?

Constructor which takes no arguments

What is a *value constructor*?

Constructor that takes arguments and sets instance variables based on the inputs

Classes

1. constructors
2. **instance variables**
3. accessors
4. modifiers
5. this keyword
6. super keyword

Instance Variables

Declared in a class but outside of any method

Instance variables belong to a specific instance of a class

What are the instance variables in Instructor.java

Classes

1. constructors
2. instance variables
3. **accessors**
4. modifiers
5. this keyword

Accessors

What are accessors?

- Also called getters
- Return the value of instance variables
- Usually named `get`**`Var`** where **Var** is the name of the variable we want to access
- What are the accessors in Instructor.java?

Classes

1. constructors
2. instance variables
3. accessors
4. **modifiers**
5. this keyword

Modifiers

- also called setters
- changes the value of instance variables
- usually named `set`**`Var`** where **Var** is the name of the variable we want to modify
- what are the modifiers in Instructor.java

Classes

1. constructors
2. instance variables
3. accessors
4. modifiers
5. **this keyword**
6. super keyword

this

`this` is a special keyword that refers to the object inside an instance method

Allows us to access other instance variables within an instance method

equals

Comparing two objects with == compares their memory addresses

Compare Strings with .equals()

- Strings are objects!

Custom equals for objects:

```
public boolean equals(Object o)
```

toString

- returns a String representation of the object
- toString.java

Static vs Non Static

Static vs Non Static

Static:

Belongs to the class rather than to any particular instance of the class

Can be invoked without the need for creating an instance of the class

Non Static (instance):

Belongs to the object (instance) of the class

Can be invoked only through an instance of the class.

Static vs Non Static

Objects can have either *static* or *instance* methods

- static methods use syntax `<ClassName>.<methodName>`
- instance methods use syntax `<object>.<methodName>`

Static

Create a class `MathUtils` with a static method `factorial(n)` that calculates the factorial of a given number `n`. Demonstrate its usage in the `main` method by calculating the factorial of a few numbers.

Inheritance

A class ***inherits*** variables and methods from an existing class.

The existing class is referred to as the **superclass** or **parent class**, and the new class is referred to as the **subclass** or **child class**.

Allows for code reuse

is a relationship

public class Bird *extends* Animal { ... }

Inheritance

```
SuperClass c = new SubClass(); //allowed
SubClass c2 = new SuperClass(); //not allowed
```

Hierarchy.java

can you extend from more than one class?

super keyword

super();

reference variable that is used to refer parent class constructors

Note:

super:

reference variable that is used to refer parent class object

Access modifiers

Specify the access-level of instance variables/methods

- public
  - code outside of the class can access the variable/method
- private
  - code outside of the class cannot access the variable/method
- protected
  - only subclasses and current class can access the variable/method

Default in java is public

In this class, make instance data private

Access Modifiers

You are developing a system to manage employees in a company. Implement the following classes according to the given specifications:

1. Create an `Employee` class with the following methods:
   - `void introduce()` - Prints "Hello, I am an employee."

2. Extend the `Employee` class to create a `Manager` class with the following additional method:
   - `void manage()` - Prints "I am managing tasks."
   - Only Manager and Executive should be able to call manage()

3. Extend the `Manager` class to create an `Executive` class with the following additional method:
   - `void makeDecisions()` - Prints "I am making decisions for the company."

Access Modifiers

Can a protected method in the superclass be called from a public method in the subclass?

Mini Example: Foo, M, and Bar

Object Oriented Programming

You are tasked with developing a program to manage fruits in a grocery store.

1. Fruit Class:

- Instance variables: `name`, `color`: Represents the color of the fruit.
- Constructor: value constructor

2. Apple Class:

- Additional instance variable: `type` (String)
- Constructor: value constructor
- Implement the `equals` method inherited from the `Fruit` class to compare apples based on their `name`, `color`, and `type`.

3. Banana Class:

- Additional instance variable: `length` (double)
- Constructor: value constructor
- Implement the `equals` method inherited from the `Fruit` class to compare bananas based on their `name`, `color`, and `length`.

Polymorphism

What is polymorphism?

Program can treat all objects that extend a base class the same

Polymorphism

Develop a Java program that demonstrates polymorphism using musical instruments.

Create a superclass called `Instrument` with a method `play()` that simply prints "Playing an instrument".

Then, create two subclasses: `Guitar` and `Piano`, each overriding the `play()` method to print "Strumming a guitar" and "Playing a piano" respectively.

In the `main()` method of your program, create an array of `Instrument` objects containing instances of both `Guitar` and `Piano`. Iterate through the array and call the `play()` method for each object.

Arrays of Objects

Problem: Implementing a Movie Database

Task 1: Define a Java class named `Movie` with the following specifications:

- The class should have private instance variables for `title`, `director`, `genre`, and `year`.
- Implement a constructor that takes parameters for initializing all instance variables.
- Implement getter methods for all instance variables.
- `displayDetails()` that prints out all the details of the movie.

Task 2: Define a Java class named `MovieDatabase` with the following specifications:

- The class should have a private instance variable to store an array of `Movie` objects.
- Implement a constructor that takes an integer parameter `size` to initialize the array size.
- `addMovie(Movie m)` add a new `Movie` object to the database.
- `searchByTitle(String title)` prints out details of the movie with matching title, if found.
- `searchByDirector(String director)` prints out details of all movies directed by the specified director, if any.
- `displayAllMovies()` that prints out details of all movies in the database

Interfaces

- An interface is a contract - A set of shared methods that users **must** implement
- create a program to calculate the area of different shapes, such as circles, rectangles, triangles etc.
- For each shape, you should be able to print the shape name and area
- Every time someone adds a new shape, they **must** include the methods for getName() and getArea()

Interfaces

- For any new shape that is created, we want to **enforce** that these methods are also implemented.

```
interface Shape {
    public double getArea();
    public String getName();
}
```

```
class Circle implements Shape {
```

Interfaces

A contract - A set of shared methods that users **must** implement

A collection of method signatures with no bodies

A class can implement more than one interface

Interfaces

An interface is not a class!

A class is what an object **is**

An interface is what an object **does**

- can not be instantiated
- no constructors
- incomplete methods

Interfaces

Example: implement the `Building` interface

Interfaces

Make a `Hammer` class and a `Screwdriver` class which implement two interfaces: `Maintainable` and `Usable`

Inheritance vs Interfaces

Each of these lines is related to either interfaces or inheritance...

- `extends` keyword
- guarantees a class has implemented certain methods
- `implements` keyword
- reuses implementations
- *is-a* relationship
- specifies what a class *does*

Exceptions

What is an exception?

What are some exceptions you have encountered?

Exceptions

Handling exceptions

```
try {
   some code where errors may occur
} catch (<some exception> <variable>) {
   some corrective action you could perform
}
```

Exceptions

Write code to:

1. Initialize an array
2. Ask the user for an input index
3. If the index is out of bounds print “oh no!”
    a. make sure an exception is not thrown

Exceptions

Write code to:

1. Ask the user for two numbers
2. If the index is out of bounds print “oh no!”
    a. make sure an exception is not thrown

try-catch

TryCatchExample.java

What will be printed?

try-catch

1. Read data from a text file named "input.txt".
2. Read each line from the file and process it according to the following rules:
   - If the line starts with "ADD", extract the following number and add it to a running total.
   - If the line starts with "SUB", extract the following number and subtract it from the running total.
   - If the line starts with "MUL", extract the following number and multiply it with the running total.
   - If the line starts with "DIV", extract the following number and divide the running total by it (handle division by zero gracefully).
   - Ignore any lines that do not conform to the above rules.
3. Display the final result of the operations performed on the data.