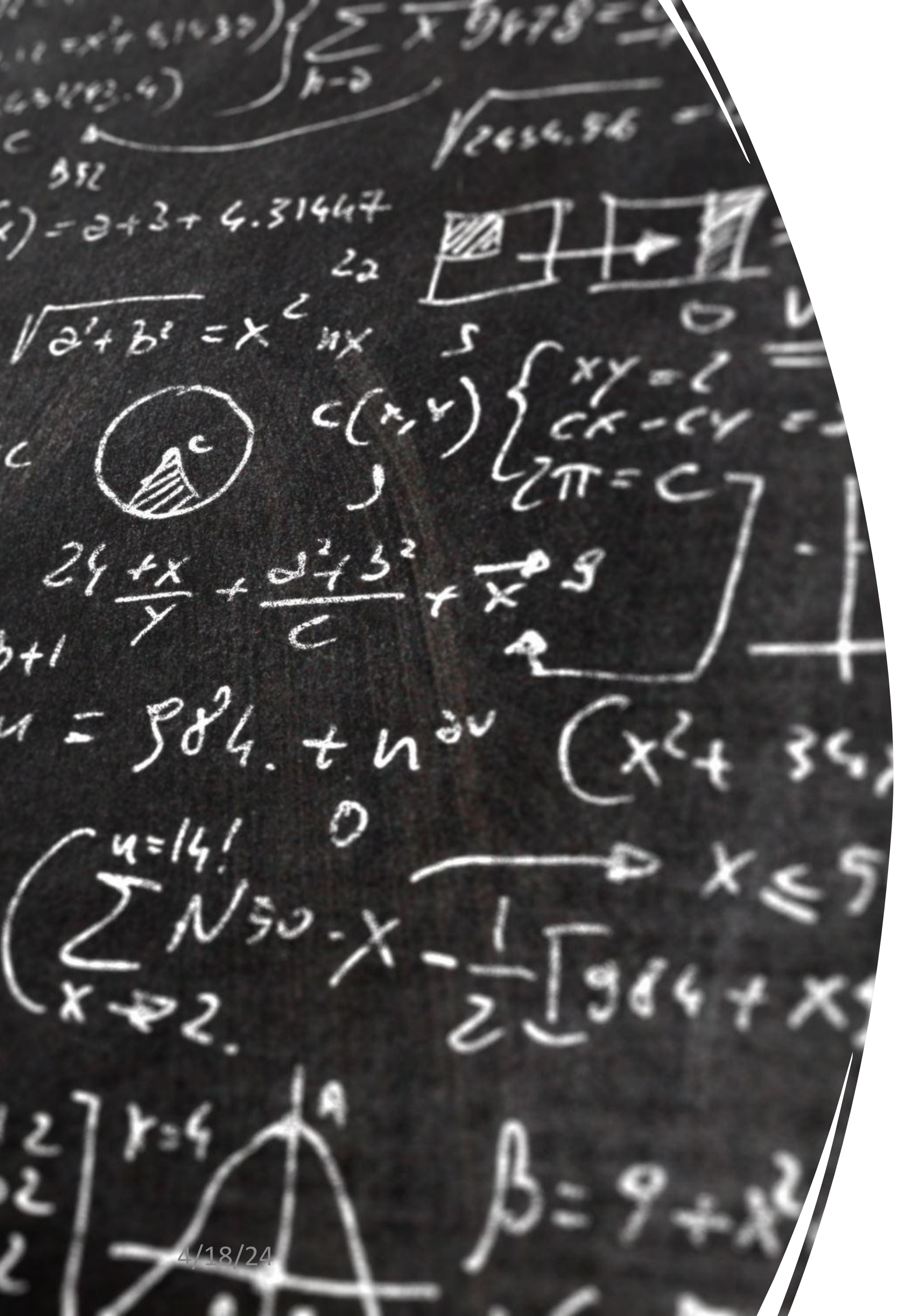

# CS 113 – Computer Science I

# Lecture 23 – Midterm 2 Review

Thursday 4/18/2024

## Announcements

Midterm next week (Tuesday April 23rd)

HW8 and Lab8 due tonight

No lab today - 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

## 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 *<u>extends</u>* 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."

1. 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()

1. 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 <u>a contract </u>- 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 **<u>is</u>**

An interface is what an object **<u>does</u>**

- 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.