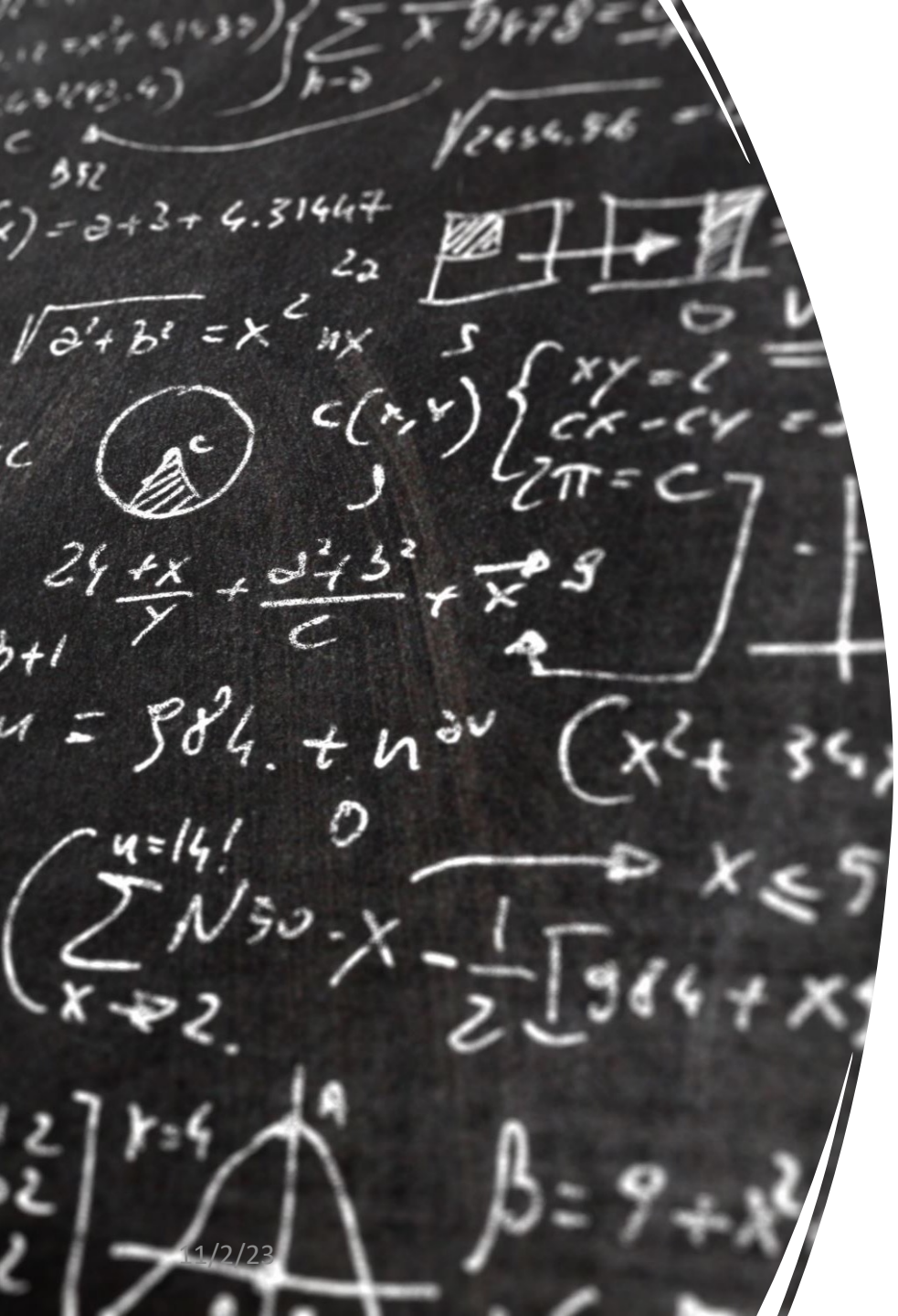


CS 113 – Computer Science I

Lecture 16 – Class Design

Tuesday 11/02/2023

Announcements

HW 06 – Due Monday 11/06

- Loops

HW07 – Due Monday 11/13

Board game

longer one

Lab06 and Lab07 are relevant

Agenda

- Objects & Classes
- Mutability

Class

A blueprint for a custom data type

A template for how data/information is stored

Contains a set of methods for how to interact/operate on the stored data

Classes and objects

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

An **object** is to a **class** as a

cat is to an **animal**

tulip is to an **flower**

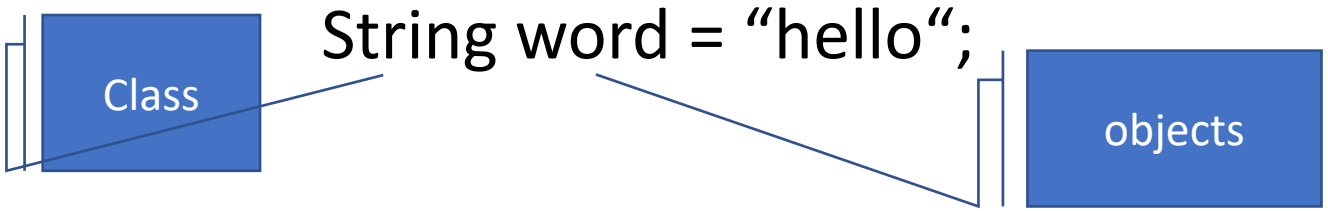
cookie it to a **snack**

Socrates is to a **human**

Classes and objects

A **class** defines the characteristics of a type (data and methods)

An **object** is a particular example of a class



Java is a strict object-oriented programming language, meaning all code must be inside a class!

Using objects: some special methods

The **constructor method** is called when you do a `new`

accesors (aka getters)
return the values of instance variables

mutators (aka setters)
set the values of instance variables

toString()
returns a string representation of an object

Defining classes

By defining our own classes, we can create our own data types

A class definition contains

- the data contained by the new type (**instance variables**)

- the operations supported by the new type (**instance methods**)

Object-oriented programming (OOP)

Method for designing programs in terms of objects

Recall: Top-down design

- the “nouns” in your feature list correspond to classes/data
- the “verbs” correspond to methods

Bank example

Keep track of account balances

Use an array:

- Each index represents another account
- The value represents the account's balance

Determine how many accounts we can hold:

- Create a new array of fixed size

Defining the Bank class

```
public class Bank {
    private int size;
    private String name;
    private String[] clients;
    private double[] accounts;

    public Bank(String bankName, int numClients) {
        name = bankName;
        size = numClients;
        clients = new String[size];
        accounts = new double[size];

    }
    public String getName() {
        return name;
    }

}
```

Testing the Bank class

```
public static void main(String args[])
{
    Bank boa = new Bank(”Bank of America", 10);
    System.out.println(”Bank: "+boa.getName());
}
```

Bank Actions

How can we find out how much money the bank is holding at once?

How can we find out which account is currently overdraft?

What other questions might the bank want to know?

Exercise: Bank Account

BankAccount should have the following data:

- Name
- Amount

BankAccount should have the following operations:

- currentBalance() // returns current amount in the bank account
- withdraw(float amt) // withdraw the given amount from the account
- deposit(float amt) // deposit the given amount to the account

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

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

Default in java is `public`

Access modifiers

Default in java is `public`

In this class, make instance data private

OOP Example & Design: Vending machine

OOP Example: Snack

Name

Cost

Quantity

Defining the snack class

```
public class Snack {
    private int mQuantity;
    private double mCost;
    private String mName;

    public Snack(String name, int quantity, double cost) {
        mQuantity = quantity;
        mCost = cost;
        mName = name;
    }
    public String getName() {
        return mName;
    }

    public void buy() {
       if (mQuantity > 0) {
           mQuantity--;
       }
    }
}
```

Testing the Snack class

```
public static void main(String args[])
{
    Snack snack = new Snack("Slurm", 10, 1.5);
    System.out.println("Snack: "+snack.getName());
}
```

Agenda

- Objects & Classes
- **Mutability**

Mutable vs Immutable

Mutable:

Values can change

Immutable:

Values cannot change

Strings and Integers are immutable