

LEC 01

**CSE 122**

# Java Review & Functional Decomposition

Questions during Class?

Raise hand or send here

sli.do #cse122




BEFORE WE START

*Talk to your neighbors:  
What's your favorite vintage  
computer game?*

Music: [122 24sp Lecture Tunes](#) 

Instructor	Miya Natsuhara & Kasey Champion			
TAs	Ayush Poojitha Chloe Ailsa Jasmine Lucas Logan Kyle	Jacob Atharva Rucha Megana Eesha Zane Colin Ronald	Saivi Shivani Kavya Steven Ken Chaafen Smriti Ambika	Elizabeth Aishah Minh Katharine


# Lecture Outline

- **Announcements/Reminders** 
- Review Java
- Functional Decomposition
- Code Quality
- First Assignment
  - Grading

# Announcements

- Hope you had fun in your first quiz section yesterday!
- Creative Project 0 (C0) released later today, due next Thursday, April 4th
  - Focused on Java Review + Functional Decomposition
- Reminder: Java Review Session Monday, April 1st
  - Monday 12:30am-1:20pm (SIEG 134)
- IPL will also open on Monday!
- Sign up for Kasey 1:1 time: [bit.ly/kasey1on1s](https://bit.ly/kasey1on1s)

# Reminders

- Fill out the [Introductory Survey](#)
-  Complete the pre-class material (PCM) for Wednesday (see calendar)
  - Will be posted after class today
- Attend quiz section on Tuesday!

# Lecture Outline

- Announcements/Reminders
- **Review Java** ◀
- Functional Decomposition
- Code Quality
- First Assignment
  - Grading

# Reminders: Review Java Syntax

[Java Tutorial](#) reviews all the relevant programming features you should familiar with (even if you don't know them in Java).

- Printing and comments
- Variables, types, expressions
- Conditionals (if/else if/ else)
- Loops (for and while)
- Strings
- Methods
- Parameters & Returns
- User input (Scanner)
- Arrays(1 and 2 dimensional)

# Hello World

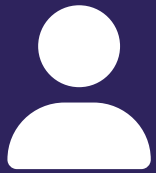
- Java Specifics

- Every program needs a **class**
- Runnable programs need a **main** method (*signature* must exactly match)
- **System.out.println** to print
- **"Hello world"** is a **String**

```
public class HelloDemo {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```

- Running on [Ed](#)

- **Run** runs your program
- **Mark** submits and runs autograder
  - Submit as many times as you like
  - “Shotgun submission” = Unhelpful habit
- **Solution** shows solution (if applicable)



# Practice : Think



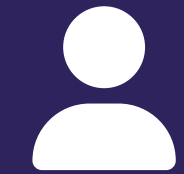
sli.do

#cse122

## In-Class Activities

- **Goal:** Get you actively participating in your learning
- Typical Activity
  - Question is posed
  - **Think** (1 min): Think about the question on your own
  - **Pair** (2 min): Talk with your neighbor to discuss question
    - If you arrive at different conclusions, discuss your logic and figure out why you differ!
    - If you arrived at the same conclusion, discuss why the other answers might be wrong!
  - **Share** (1 min): We discuss the conclusions as a class
- During each of the **Think** and **Pair** stages, you will respond to the question via a sli.do poll
  - Not worth any points, just here to help you learn! 😊





# Practice : Think

[sli.do](https://sli.do)

#cse122

## What is the output of this Java program?

```
public class Demo {
    public static void main(String[] args) {
        int[] nums = {1, 4, 4, 8, 13};

        int totalDiff = 0;
        for (int i = 1; i <= nums.length; i++) {
            totalDiff += (nums[i] - nums[i - 1]);
        }
        System.out.println("Total Diff = " + totalDiff);
    }
}
```

- A) Total Diff = 12
- B) Total Diff = 10
- C) Total Diff = 9
- D) Exception!



# Practice : Pair



sli.do #cse122

## What is the output of this Java program?

```
public class Demo {  
    public static void main(String[] args) {  
        int[] nums = {1, 4, 4, 8, 13};  
  
        int totalDiff = 0;  
        for (int i = 1; i <= nums.length; i++) {  
            totalDiff += (nums[i] - nums[i - 1]);  
        }  
        System.out.println("Total Diff = " + totalDiff);  
    }  
}
```

- A) Total Diff = 12
- B) Total Diff = 10
- C) Total Diff = 9
- D) Exception!

# Case Study: Minesweeper

Minesweeper is a classic puzzle game that involves a grid of squares, some of which contain hidden mines. The objective of the game is to uncover all the squares that do not contain mines, without detonating any of the mines.

In the board that the player is working from, each square contains either:

- A mine
- A number indicating how many mines are adjacent to the square
- Is blank (indicating there are 0 mines adjacent to the square)

The player clicks on a square to reveal what is hidden underneath. If the square contains a mine, the game is over and the player loses. If not, the player uses the information in that square to determine how to proceed.

Using logic and deduction, the player must determine the locations of the mines and mark them with flags. Once all the mines have been flagged, the game is won. The game comes with different levels of difficulty which may determine things in the game such as the size of the grid or the number of mines hidden within it.




# Case Study: Minesweeper

```
Welcome to MineSweeper!
```

```
Please choose game difficulty (NOTE: Difficulty: 1 to 10): 3
```

```
- - - - - - - - - - - - -  
0 0 1 X 1 1 X 1 0 0 0 0 0  
0 0 1 1 1 1 1 1 0 0 0 0 0  
1 1 1 0 0 0 0 0 0 0 0 0 0  
1 X 2 1 1 0 0 0 0 0 0 0 0  
1 1 2 X 1 0 0 1 1 1 0 0 0  
0 0 1 1 1 0 0 1 X 1 0 0 0  
0 0 0 0 0 0 0 1 1 1 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 0 0 0 0 0 0 0 0 0 0  
1 X 1 1 1 1 0 0 0 0 0 0 0  
1 1 1 1 X 2 1 1 0 0 0 0 0  
1 1 0 1 1 2 X 1 0 0 0 0 0  
X 1 0 0 0 1 1 1 0 0 0 0 0  
- - - - - - - - - - - - -
```

# Lecture Outline

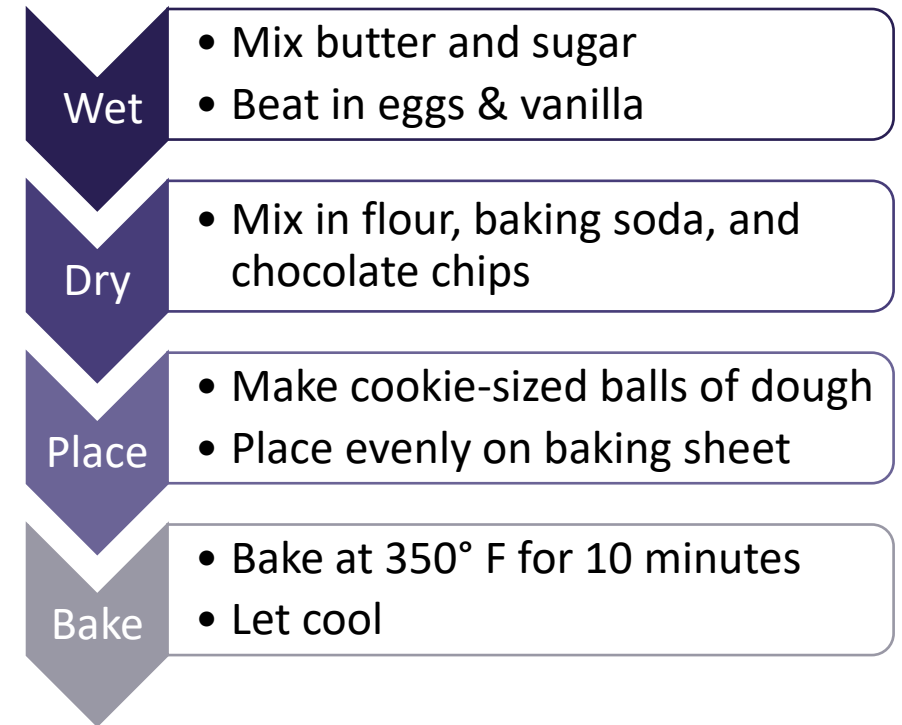
- Announcements/Reminders
- Review Java
- **Functional Decomposition** 
- Code Quality
- First Assignment
  - Grading

# Functional Decomposition

**Functional decomposition** is the process of breaking down a complex problem or system into parts that are easier to *conceive, understand, program, and maintain*.

## “Bake the cookies”

- Mix butter and sugar
- Beat in eggs & vanilla
- Mix in flour, baking soda, and chocolate chips
- Make cookie-sized balls of dough
- Place evenly on baking sheet
- Bake at 350° F for 10 minutes
- Let cool



# Functional Decomposition

In our code, functional decomposition often means breaking a task into smaller methods (also called functions).

## Example: Minesweeper

- Print intro
- Ask for difficulty from user
- Set up blank game board
- Randomly decide where to place mines
- Place counts in each non-mine square

# Avoid Trivial Methods

Introduce methods to decompose a complex problem, not just for the sake of adding a method.

## Bad example:

```
public static void printMessage(String message) {  
    System.out.println(message);  
}
```

## Good Example:

```
public static double round(double num) {  
    return ((int) Math.round(num * 10)) / 10.0;  
}
```

Rule of thumb: A method should do at least two steps

- Ask yourself: Does adding this method make my code easier to understand?



# Lecture Outline

- Announcements/Reminders
- Review Java
- Functional Decomposition
- **Code Quality** ◀
- First Assignment
  - Grading

# Code Quality

“Programs are meant to be read by humans and only incidentally for computers to execute.” – Abelson & Sussman, SICP

Code is about *communication*. Writing code with good **code quality** is important to communicate effectively.

Different organizations have different *standards* for code quality.

- Doesn't mean any one standard is wrong! (e.g., APA, MLA, Chicago, IEEE, ...)
- Consistency is very helpful within a group project
- See our [Code Quality Guide](#) for the standards we will all use in CSE 122

# CSE 122 Code Quality

Examples relevant for this week

- Naming conventions
- Descriptive variable names
- Indentation
- Long lines
- Spacing
- Good method decomposition
- Writing documentation



# Practice : Pair

**What does this code do? How could you improve the quality of this code? (No Slido poll)**

```
public static int l(String a,char b){
    int j=-1;for(int a1=0;a1<a.length();a1  ++) {
    if (a.charAt(a1) == b) {
        j = a1;
    } } if(j==-1){return -1;} else {
return j;} }
```




# Practice : Pair

**What does this code do? How could you improve the quality of this code? (No Slido poll)**

```
public static int lastCharInString(String str, char c) {
    int loc = -1;
    for(int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == c) {
            loc = i;
        }
    }
    return loc;
}
```

# Lecture Outline

- Announcements/Reminders
- Review Java
- Functional Decomposition
- Code Quality
- **First Assignment** 
  - Grading

# Graded Course Components

- Your grade will consist of the following categories:
- Each mark is graded on the scale:
  - **E**(xcellent)
  - **S**(atisfactory)
  - **N**(ot yet)

Category	#	Marks per	Total Marks
Programming Assignments	4	4 ( <a href="#">Behavior, Concepts, Quality, Testing/Reflection</a> )	16
Creative Projects	4	1	4
Quizzes	3	3 (3 questions)	9
Exam	1	6 (6 questions)	6

# Course Grades

Instead of curving the class, we'll use a bucket system:

- Marks earned place in an initial bucket, additional S+ marks improve grade.
- Must meet all requirements of a bucket for initial placement.
- These are minimum GPA guarantees – grade can always be higher than minimum promise. 😊

Minimum Grade	Required S+	Required E
3.5	30	27
3.0	27	22
2.5	24	17
2.0	21	0
1.5	14	0
0.7	8	0



# Creative Project 0

- Released today, due next Thursday (April 4) at 11:59 pm on Ed
  - Can hit the "Submit" button multiple times – we will grade the last submission made before the initial deadline.
  - Build good habits: Don't "shotgun debug"
  - While you can resubmit this assignment, it's important to meet due date to get as much feedback as possible.
- Focused on reviewing Java concepts and Functional Decomposition
- See [Grading Rubric](#) to know what to expect
- IPL opens Monday!