BEFORE WE START

Talk to your neighbors:

Debrief Quiz 2. How do you feel like it went in comparison to Quiz 1?

Music: 123 24su Lecture Tunes

Instructor: Joe Spaniac
TAs: Andras, Daniel, Eric, Nicole, Sahej, Trien, Zach
Lecture Outline

• Announcements

• Recursion
  - Definition
  - Call Stack

• Math Examples

• Revisiting Reflections
Announcements

• Quiz 2 Completed! 😊➡️
  - Congrats! Expect grades back around next Thursday (hopefully)
  - Practice metacognition: how did that go? What can you learn about your studying process and how can you incorporate it before the next quiz?

• Programming Assignment 2 due in one week (7/24) @ 11:59pm
  - Already said this before – it’s hard, start it sooner rather than later

• C2 / R2 grades out after lecture

• Resubmission Period 3 closes this Friday (7/19) @ 11:59pm
  - Available assignments: C1, P1, C2
  - Last opportunity to resubmit C1

• Reminder: Grade guarantee calculator
Lecture Outline

• Announcements

• **Recursion**
  - Definition
  - Call Stack

• Math Examples

• Revisiting Reflections
Recursion

“The repeated application of a recursive procedure or definition”
- Oxford Languages

• Real-world definition: defining a problem in terms of itself
  - Case in point: above definition
  - Further natural examples:
Recursion

“The repeated application of a recursive procedure or definition”
- Oxford Languages

• Real-world definition: defining a problem in terms of itself
  - Case in point: above definition
  - Further natural examples:
Recursion

“The repeated application of a recursive procedure or definition”
- Oxford Languages

• Real-world definition: defining a problem in terms of itself
  - Case in point: above definition

• Computer science definition: when a method calls itself
  - “Alternative” to iteration (can combine for powerful results)

• 😞 Wouldn’t that just lead to an infinite loop?
  - Yes! If you do things wrong...
  - Let’s review how method calls work
Method Calls

• Regardless how you use them, methods work the same way!
  - Pause execution, finish method, return where you left off

• How does Java keep track of the prior method (LIFO)?
  - Something called the Call Stack
Call Stack

public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
Call Stack

```
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
```

Console:
Call Stack

public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
Call Stack

```
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
```
public static void main(String[] args) {
    mystery1();
    mystery2();
}

class Call Stack {
    public static void mystery1() {
        System.out.println("One");
    };
    public static void mystery2() {
        System.out.println("Two");
        mystery3();
    }
    public static void mystery3() {
        System.out.println("Three");
    }
}
Call Stack

public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println("One");
}

public static void mystery2() {
    System.out.println("Two");
    mystery3();
}

public static void mystery3() {
    System.out.println("Three");
}
public static void main(String[] args) {
    mystery1();
    mystery2();
}

public static void mystery1() {
    System.out.println(“One”);
}

public static void mystery2() {
    System.out.println(“Two”);
    mystery3();
    System.out.println(“Three”);
}

public static void mystery3() {
    System.out.println(“Three”);
}
Method Calls

• Regardless how you use them, methods work the same way!
  - Pause execution, finish method, return where you left off

• How does Java keep track of the prior method (LIFO)?
  - Something called the Call Stack

🤔 Wouldn’t that just lead to an infinite loop?

```java
public static void recursion() {
    System.out.println("Woah");
    recursion();
}
```
Method Calls

• Regardless how you use them, methods work the same way!
  - Pause execution, finish method, return where you left off

• How does Java keep track of the prior method (LIFO)?
  - Something called the Call Stack

🤔 Wouldn’t that just lead to an infinite loop?
  - Yes! We get something called a StackOverflowException

• How do we avoid infinite recursion?
Recursive Methods

• 2 components of every recursive method:

• Recursive case
  - Decompose problem into subproblem
  - Make the actual recursive call
  - Combine results meaningfully

• Base case
  - Simplest version of the problem
  - No subproblems to break into
  - Return known answer
Recursive Methods

• 2 components of every recursive method:

• Recursive case
  - Decompose problem into subproblem
  - Make the actual recursive call
  - Combine results meaningfully

• Base case
  - Simplest version of the problem
  - No subproblems to break into
  - Return known answer

If decomposing moves you closer to the base, no infinite recursion!
Lecture Outline

• Announcements

• Recursion
  - Definition
  - Call Stack

• Math Examples

• Revisiting Reflections
Math Examples

\( n! / \text{factorial}(n) = \text{product of all positive integers} \leq n \)

• Two ways of viewing this idea:

• \( n! = n \times (n - 1) \times (n - 2) \times \cdots \times 2 \times 1 \)
  - Iterative approach - loop through all values and multiply together

• \( n! = n \times (n - 1)! \)
  - Recursive approach – decompose into subproblem and combine
  - What would our base case / simplest input (\( n \)) be?
Math Examples

\( n! / \text{factorial}(n) = \text{product of all positive integers } \leq n \)

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example 3!

4! = 4 \times 3!
3! = 3 \times 2!
2! = 2 \times 1!
1! = 1 \times 0!
0! = 1
Math Examples

\[ n! = \text{factorial}(n) = \text{product of all positive integers} \leq n \]

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example \(3!\):

\[
\begin{align*}
4! &= 4 \times 3! \\
3! &= 3 \times 2! \\
2! &= 2 \times 1! \\
1! &= 1 \times 0! \\
0! &= 1
\end{align*}
\]
Math Examples

\[ n! / \text{factorial}(n) = \text{product of all positive integers} \leq n \]

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example 3!

\[
\begin{align*}
4! &= 4 \times 3! \\
3! &= 3 \times 2! \\
2! &= 2 \times 1! \\
1! &= 1 \times 1
\end{align*}
\]
Math Examples

\[ n! / \text{factorial}(n) = \text{product of all positive integers} \leq n \]

• Reminder, recursive approach: \( n! = n \times (n-1)! \)
  - Let’s trace through with a simple example 3!

\[ 4! = 4 \times 3! \]
\[ 3! = 3 \times 2! \]
\[ 2! = 2 \times 1! \]
\[ 1! = 1 \]
Math Examples

\( n! / \text{factorial}(n) = \text{product of all positive integers} \leq n \)

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example 3!

\[
4! = 4 \times 3! \\
3! = 3 \times 2! \\
2! = 2 \times 1
\]
Math Examples

\[ n! \text{ / factorial}(n) = \text{product of all positive integers} \leq n \]

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let's trace through with a simple example 3!

\[
4! = 4 \times 3! \\
3! = 3 \times 2! \\
2! = 2
\]
Math Examples

\[ n! / \text{factorial}(n) = \text{product of all positive integers } \leq n \]

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example 3!

\[
4! = 4 \times 3!
\]

\[
3! = 3 \times 2
\]
Math Examples

\( n! / \text{factorial}(n) = \text{product of all positive integers} \leq n \)

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example 3!

\[ 4! = 4 \times 3! \]
\[ 3! = 6 \]
Math Examples

\[ n! \text{ / factorial}(n) = \text{product of all positive integers} \leq n \]

- Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example \( 3! \)

\[ 4! = 4 \times 6 \]
Math Examples

\( n! / \text{factorial}(n) = \text{product of all positive integers } \leq n \)

• Reminder, recursive approach: \( n! = n \times (n - 1)! \)
  - Let’s trace through with a simple example 3!

\( 4! = 24 \)
Lecture Outline

• Announcements

• Recursion
  - Definition
  - Call Stack

• Math Examples

• Revisiting Reflections
Revisiting Reflections

• Throughout this course, we’ll ask you to form opinions on topics
  - Provide exposure to issues so you can decide for yourself

• Opinions aren’t formed in a vacuum
  - Exposure to various viewpoints reinforces/challenges perspectives
  - Shouldn’t be making arbitrary decisions
    - Rationalization is often important! (Not always necessary, but helps in communication)

• Integrating reflections to in-class components
  - Discuss opinions, challenge assumptions, potentially change minds
  - Please be respectful of other people’s opinions
    - There are no “right” or “wrong” answers to these questions
    - Everyone has different experiences with the world that informs their decisions
P1 / C2 Reflections

• P1 - Video: End to End Encryption (E2EE)
  - Q6: Do you believe it's necessary to sacrifice privacy for the "greater good" / safety of modern society? Why or why not?

• C2 - Behavioral Mapping Study
  - Describe what game you implemented / define rules if necessary
  - Q7: If you were to make one change to your implementation centered around usability, what would it be?
  - Bonus: How could you make your game more accessible for users? No formal definition, just how you interpret the question.