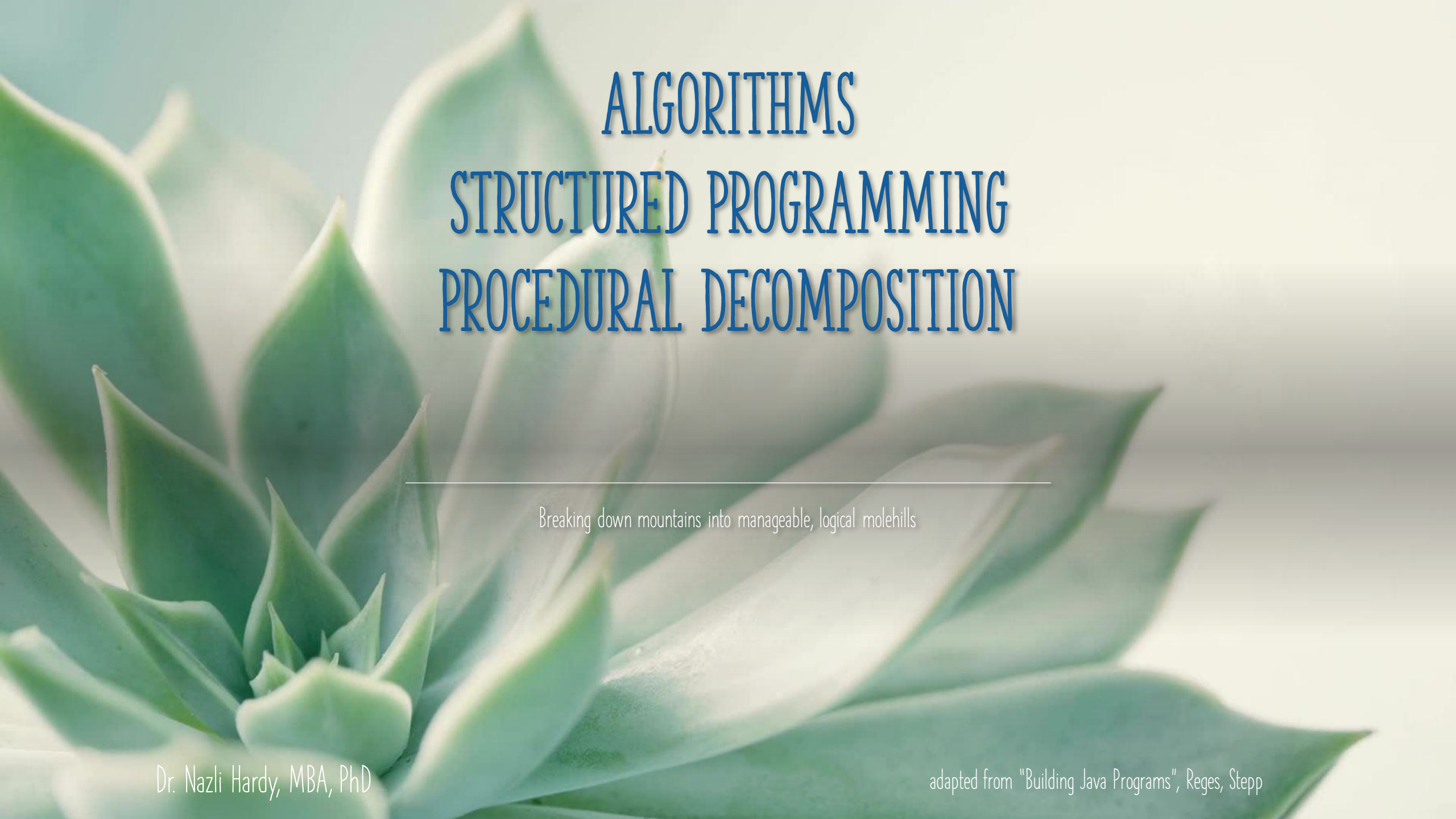

ALGORITHMS
STRUCTURED PROGRAMMING
PROCEDURAL DECOMPOSITION

Breaking down mountains into manageable, logical molehills

Dr. Nazli Hardy, MBA, PhD

adapted from "Building Java Programs", Reges, Stepp

OVERVIEW OF LECTURE

- What is an Algorithm?
- Baking a Cake
- Divide and Conquer
- Stop (in the name of efficiency)
- Static Methods
- Prince of Bel Air
- Lollipop
- More Method Calls
- FooBarBaz

A COMPLICATED TASK

- Consider the process of baking a wedding cake
  - where do we start?
  - shall we build a long cylinder and carve it into shape?

DIVIDE & CONQUER

- Programs (especially large and/or complex ones) can be sub-divided into logical sub-programs

- Advantages: [let's discuss]

PROGRAM VERSION 1

- The program works correctly
- But what's wrong with this?
- What is a better way?

```
// Author: Reges & Stepp
// This program prints assorted figures.

public class Figures1 {
    public static void main(String[] args) {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\_____/");
        System.out.println();
        System.out.println("\\         /");
        System.out.println(" \\_____/");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\_____/");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("+--------+");
    }
}

Chap1/Figures1
```

QUICK RECAP: CREATING A NEW PROJECT IN ECLIPSE

WHAT IS AN ALGORITHM & WHY IS IT USEFUL?

Algorithm:

Algorithms can be used to:

STATIC METHODS

- Static method:
  - a block of Java statements that is given a name
  - denotes the structure of the program
  - unit of procedural decomposition that eliminates redundancy by code reuse

  [recall baking the wedding cake]

- Typically we can break a class into several static methods - each of which solve some piece of the overall problem

- You have already seen a static method called _____, which is in the controlling class in a Java application

```
public class Figures1 {
    public static void _____(String[] args) {
```

- The static methods you will use follow a similar structure

```
public static void <name>( ) ….{
```

TO CREATE EFFECTIVE STATIC METHODS

- design an algorithm
  - decide what are the overall tasks
  - look for patterns
  - look at the structure and which commands are repeated
    - group these together
- declare (write down/ create) a method for each overall task
- call (run) the methods
  - note that the program's `main()` method executes the other methods to perform the overall tasks [pulls everything together]

STRUCTURING STATIC METHODS

Program Structure Using Static Methods

```
Class
  method 1
    statement 1a;
    statement 1b;
    :
    statement 1n;
          •
  method 2
    statement 2a;
    statement 2b;
    :
    statement 2n;
       •
  method 3
    statement 3a;
    statement 3b;
    :
    statement 3n;

  :
  method n
```

DECLARING A STATIC METHOD

....

...give your method a name so that it can be called/ executed/ run

```
public static void <name>()...   {
```

The keyword `public`: indicates that this method is available to be used by all parts of your program

The keyword `static`: indicates that this is a static method (i.e. does not define objects and is not an instance method)

The keyword `void`: indicates that this method executes statements but does not return any value (later methods will compute and return values)

`<name>()` : e.g. drawLine() or drawTop() or writeSpaces()

```
public static void drawLine()
```

`methodName()` the parentheses specify a list (in this case an empty list) of values to be sent to your method as input. These values are called parameters (see chap 3)

PRINCE OF BEL-AIR

Desired Output of Program

- Now this is the story all about how
- My life got flipped turned upside-down

- Now this is the story all about how
- My life got flipped turned upside-down

- Now this is the story all about how
- My life got flipped turned upside-down

LET'S THINK THIS THROUGH

What is the **pattern**?

Apply what we have **learnt so far:**

Design/ structure a program to produce the desired output

SOLUTION PROCESS: DECLARE A METHOD

GROUP WORK

- Let's create a _____ that defines the repeating lines
- Call it something descriptive, for example, displayMessage()

```
public static void displayMessage() {
        // what should go in here?
```

- Now, what should we do with this method?

SOLUTION PROCESS: CALL THE METHOD

- create the _____ called main that will then 'invoke' or 'call' the method called _____ three times

```
public static void main(String[] args) {
        // calling method(s)
```

- **Let's put it altogether**

PUTTING IT TOGETHER

```
public class FreshPrince {

    public static void main(String[] args) { // invoke or call the method
        displayMessage();
        System.out.println();
        displayMessage();
        System.out.println();
        displayMessage();


    }

    public static void displayMessage() { // define the method
        System.out.println("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
//Chap1/ FreshPrince or TwoMessages
```

WHEN TO USE AND WHEN NOT TO USE METHODS

- Place statements in a static method if the statements are:
  - related structurally and/ or
  - repeated

- You should NOT create static methods for:
  - a println statement
  - only blank lines [put blank println statements in main() method]
  - unrelated/ weakly related statements [consider splitting these into 2 smaller methods]

CRITICAL THINKING/ APPLICATION

Apply what we have learnt so far.

Look for patterns. Create and use static methods as appropriate.

Let's write a program that prints the following output to the console.

```
Lollipop, lollipop
Oh, lolli lolli lolli

Lollipop, lollipop
Oh, lolli lolli lolli

Lollipop, lollipop
Oh, lolli lolli lolli

Lollipop, lollipop
Oh, lolli lolli lolli

Call my baby lollipop
```

CRITICAL THINKING/ APPLICATION: CLASS DISCUSSION

METHOD CALL

1. How many methods are there? What is the output?
2. What if we switched the order in which the methods were called
3. Advantages of using methods?

Some notes:

Method call: a command to execute another method

When a method is called, the program's execution jumps into that method, and executes its statements

After that, it jumps back to the point where the method was called from

```
// your name
// methods for structure and redundancy


public class Figures2 {
    public static void main(String[] args) {
        drawTop();
        drawBottom();
}

    // draws redundant part that
    // looks like the top of the sign
    public static void drawTop() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
    }

    // draws redundant part that
    // looks like the bottom of the sign
    public static void drawBottom() {
        System.out.println("\\         /");
        System.out.println(" \\_______/");
    }
}
```

CRITICAL THINKING

```
public class Figures1 {
    public static void main(String[] args)
    {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("|  STOP  |");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("|  STOP  |");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("+--------+");
    }
}
```

- Let's break this redundant program to its lowest common parts
- Look for patterns
- Look for repetitions
- How many lowest common parts can you find?
- We will construct the methods using the lowest common parts

APPLY WHAT YOU HAVE LEARNT: CREATE SEPARATE METHODS

GROUP WORK

PART II

METHOD CALLS: PULLING EVERYTHING ALTOGETHER

Using methods

```
public class Figures3 {
    public static void main(String[] args) {
drawTop();
drawStop();
drawBottom();
drawBottom();
drawLine();
drawTop();
drawStop();
drawBottom();
System.out.println();
drawTop();
drawLine();
}
    // draws redundant part that looks like the top of an egg
    public static void drawTop() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
    }

    // draws redundant part that looks like the bottom of an egg
    public static void drawBottom() {
        System.out.println("\\         /");
       System.out.println(" \\_______/");
    }
  // draws a line
    public static void drawLine() {
        System.out.println("+---------+");
    }
// draws the STOP
    public static void drawStop() {
        System.out.println("|  STOP    |");
    }

}
```

Not using methods

```
public static void main(String[] args) {
    System.out.println("  _______");
    System.out.println(" /       \\");
    System.out.println("/         \\");
    System.out.println("|  STOP   |");
    System.out.println("\\         /");
    System.out.println(" \\_______/");
    System.out.println();
    System.out.println("\\         /");
    System.out.println(" \\_______/");
    System.out.println("+--------+");
    System.out.println();
    System.out.println("  _______");
    System.out.println(" /       \\");
    System.out.println("/         \\");
    System.out.println("|  STOP   |");
    System.out.println("\\         /");
    System.out.println(" \\_______/");
    System.out.println();
    System.out.println("  _______");
    System.out.println(" /       \\");
    System.out.println("/         \\");
    System.out.println("+--------+");
    }
}
```

CHANGE THE ORDER IN WHICH METHODS ARE DEFINED: WHAT HAPPENS?

```
public class Figures4 {

    // draws a line
     public static void drawLine() {
         System.out.println("+---------+");
     }

    // draws redundant part that looks like the top of an egg
     public static void drawTop() {
           System.out.println("  _____");
           System.out.println(" /_____\\");
           System.out.println("/       \\");
     }

    // draws the STOP
     public static void drawStop() {
         System.out.println("|  STOP    |");
     }
     // draws redundant part that looks like the bottom of an egg
     public static void drawBottom() {
           System.out.println("\\       /");
            System.out.println("\\_____/");
     }

    public static void main(String[] args) {
            drawTop();
            drawStop();
            drawBottom();
            drawBottom();
            drawLine();
            drawTop();
            drawStop();
            drawBottom();
            System.out.println();
            drawTop();
            drawLine();
      }

}
```

CHANGE THE ORDER IN WHICH THE METHODS ARE CALLED.
WHAT HAPPENS?

```
public class Figures5 {
// draws the top of an egg
    public static void drawTop() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
    }
// draws the bottom of an egg
    public static void drawBottom() {
        System.out.println("\\         /");
       System.out.println(" \\_______/");
    }
    // draws a line
    public static void drawLine() {
        System.out.println("+---------+");
    }
    // draws the STOP
    public static void drawStop() {
        System.out.println("| STOP |");
    }
    public static void main(String[] args) {
      drawBottom();
       drawStop();
       drawBottom();
       drawTop();
       drawLine();
       drawTop();
       drawStop();
       drawBottom();
       System.out.println();
       drawTop();
       drawLine();
       drawTop();
      drawLine();
    }   }
```

METHODS THAT CALL OTHER METHODS: WHAT IS THE OUTPUT?

/ Student
// Info

```
public class FooBarBazMumble {

    public static void main(String[] args) {
        foo();
        bar();
    }
    public static void foo() {
        System.out.println("foo");
        mumble();
        System.out.println();
    }
    public static void bar() {
    System.out.println("bar");
    baz();
   }
   public static void baz() {
   System.out.println("baz");
   mumble();
   }
   public static void mumble() {
   System.out.println("mumble");
   }
}
```

- How many and which are the methods?
- Which are the statements?
- Which methods call other methods?
- What is the output?
- Order does matter - in what way?
- In what way does order not matter?

foo: http://catb.org/~esr/jargon/html/frames.html

CRITICAL THINKING: STUDENT EXERCISE

How should we approach creating this rocket, using methods?? What is the pattern? Repetitions?