CS 106A, Lecture 26
Final Exam Review 2

Plan for today

- Announcements
- HashMaps
- Classes
- Interactors
- Final Exam Tips

Plan for today

- Announcements
- HashMaps
- Classes
- Interactors
- Final Exam Tips

Announcements

- Final Exam OAE Accommodations
- Midterm Regrade Requests
- End-Quarter LaIR and Office Hours
- Graphics Contest (*drum roll *)

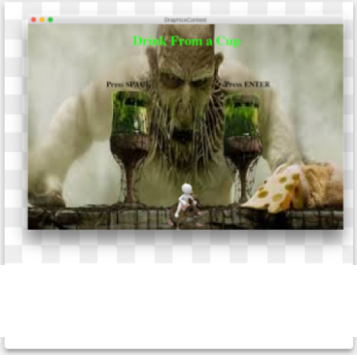

THE
OSCARS
Karels

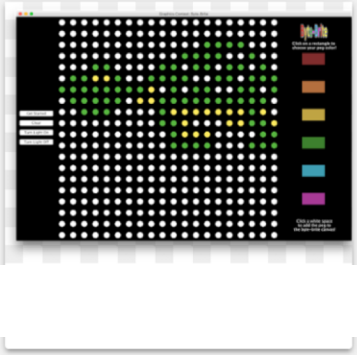

DRACO DORMIENS NUNQUAM TITILLANDUS

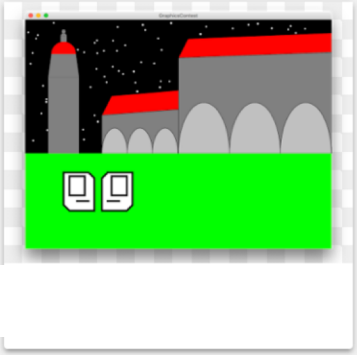

Medieval Tavern

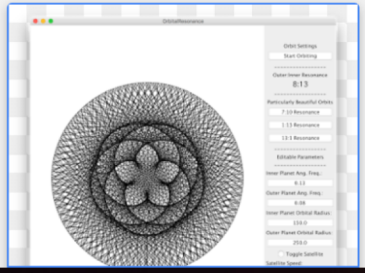

MEDIUM
Solution
Clear


Current Song: Twinkle Twinkle Little Star
Duration: 28s
CS 106Pianis

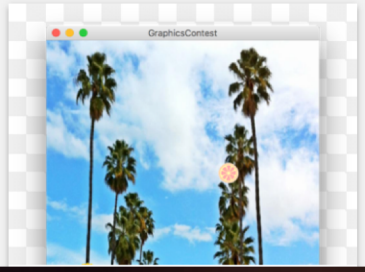

Drink From a Cup

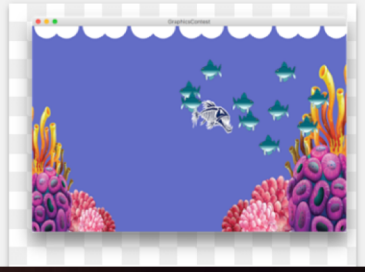


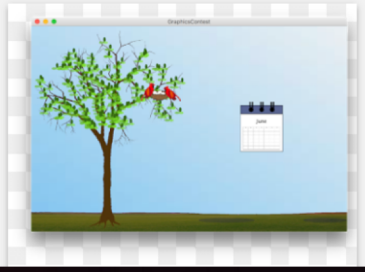




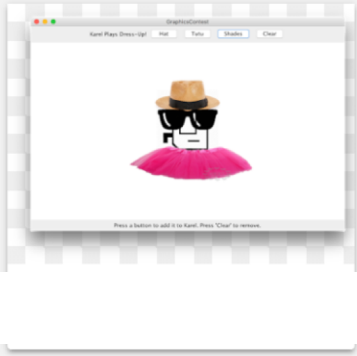

USA Score:
Canada Score:

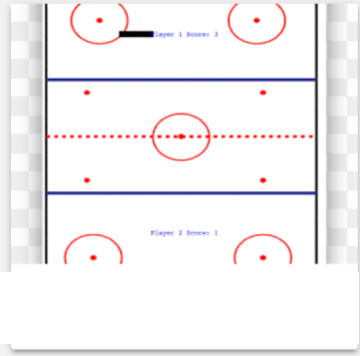

GraphicsContest



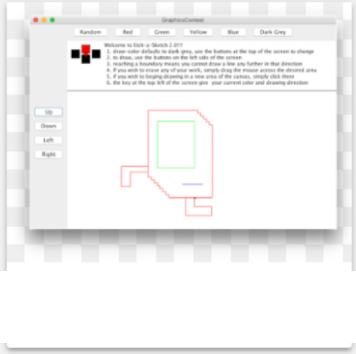


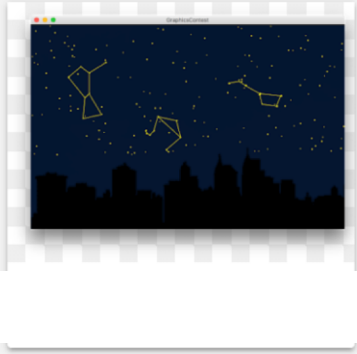

RECORD
DRAW
RESET
SONOCHROMATIC ART



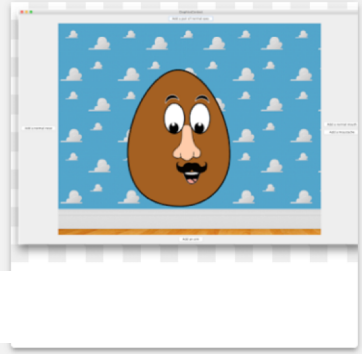

Player 1 Scores: 3
Player 2 Scores: 1

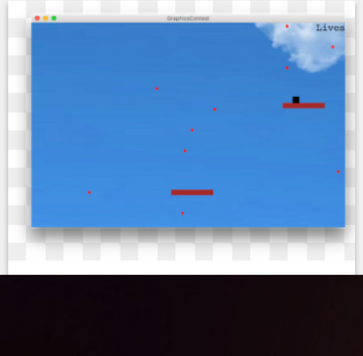

BOOM!

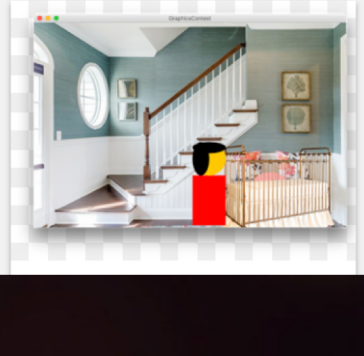


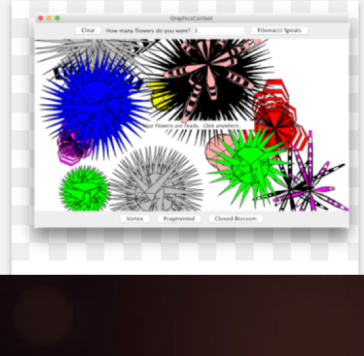


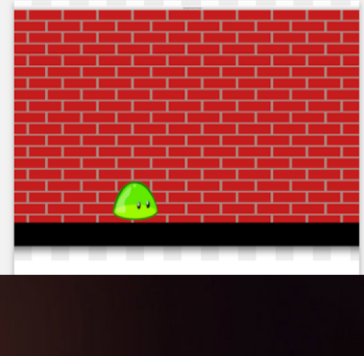

Merry Christmas, Chris!
From: Nick

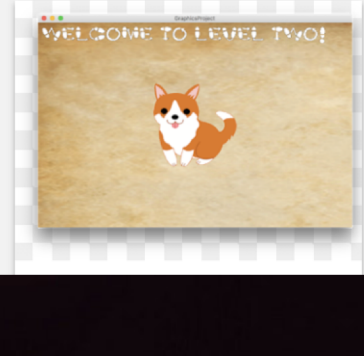



Lives

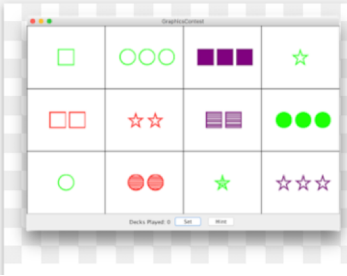


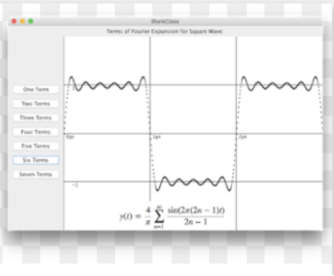


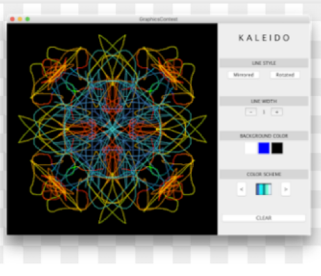


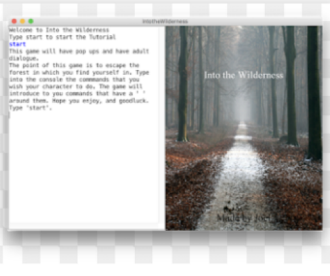

WELCOME TO LEVEL TWO!

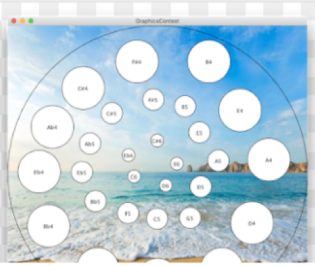

Press the spacebar to stay afloat!



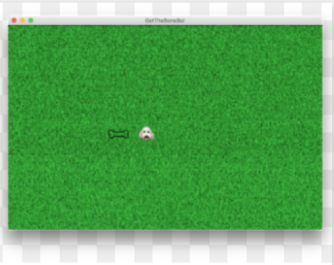

Terms of Fourier Expansion for Square Wave:

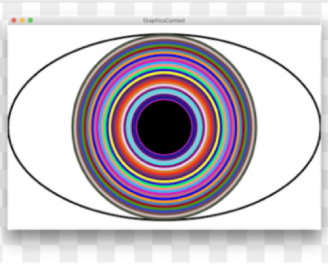

KALEIDO

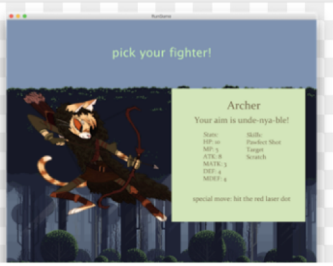

Into the Wilderness

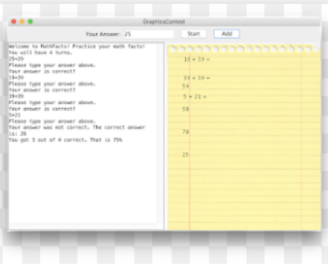


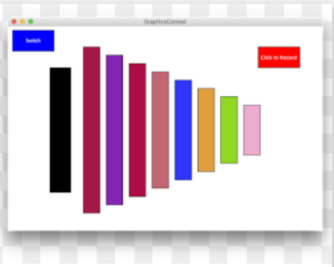

Welcome to SPREAD

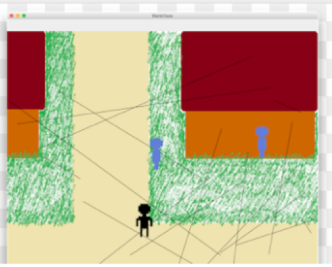


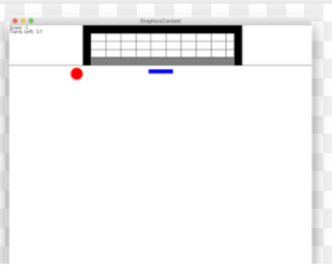


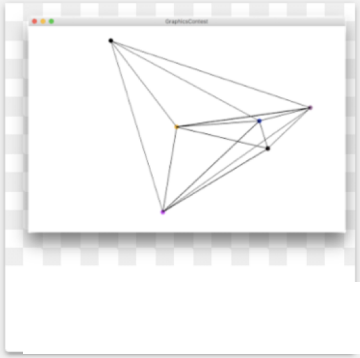

pick your fighter!
Archer
Your aim is unde-nya-ble!

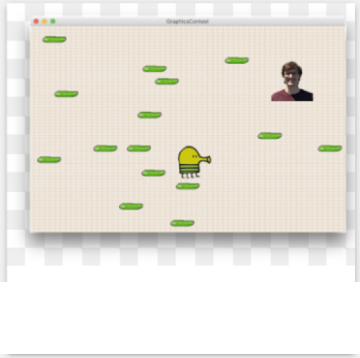


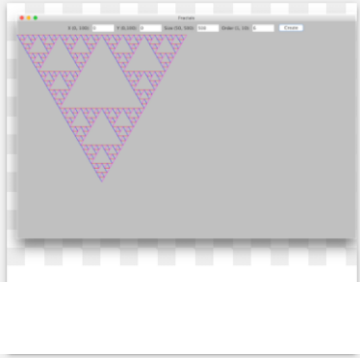

Switch
Click to Restart

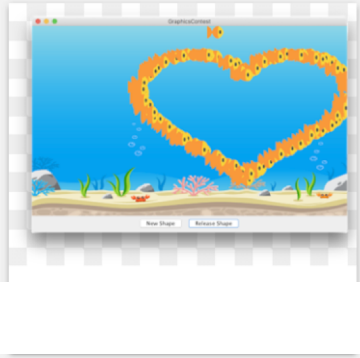


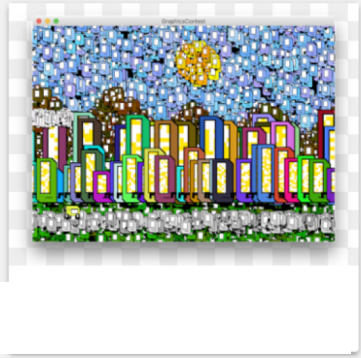


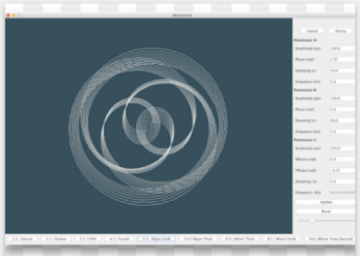


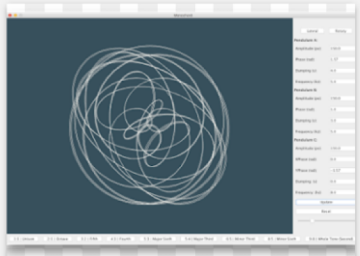


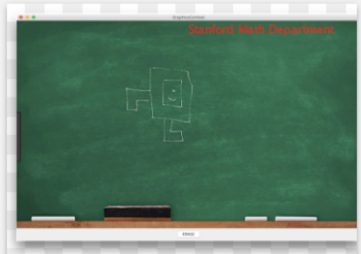


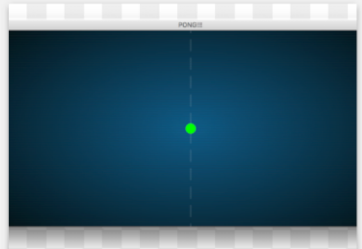


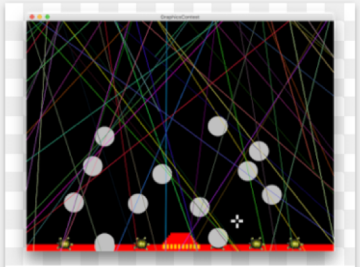




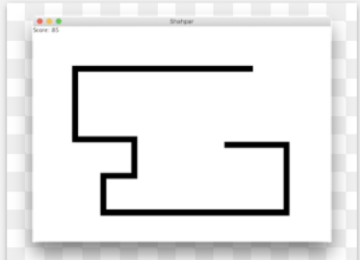


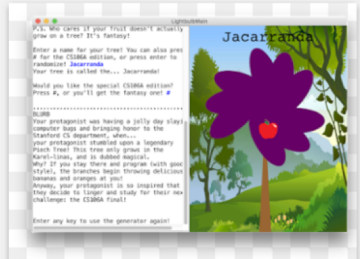

Stanford Math Department

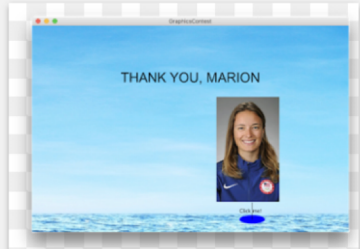





LEVEL 1
Gryffindor Quidditch Training
SCORE: 20

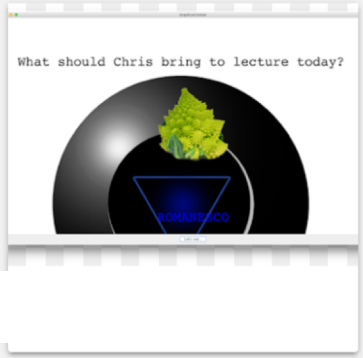


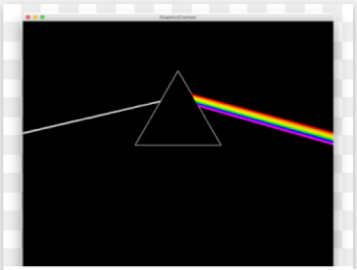

Jacarranda

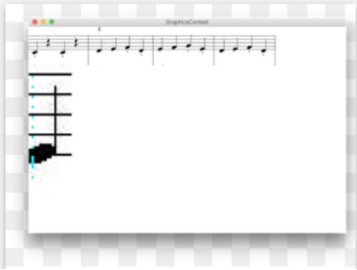

THANK YOU, MARION


CS 106A Road Racer

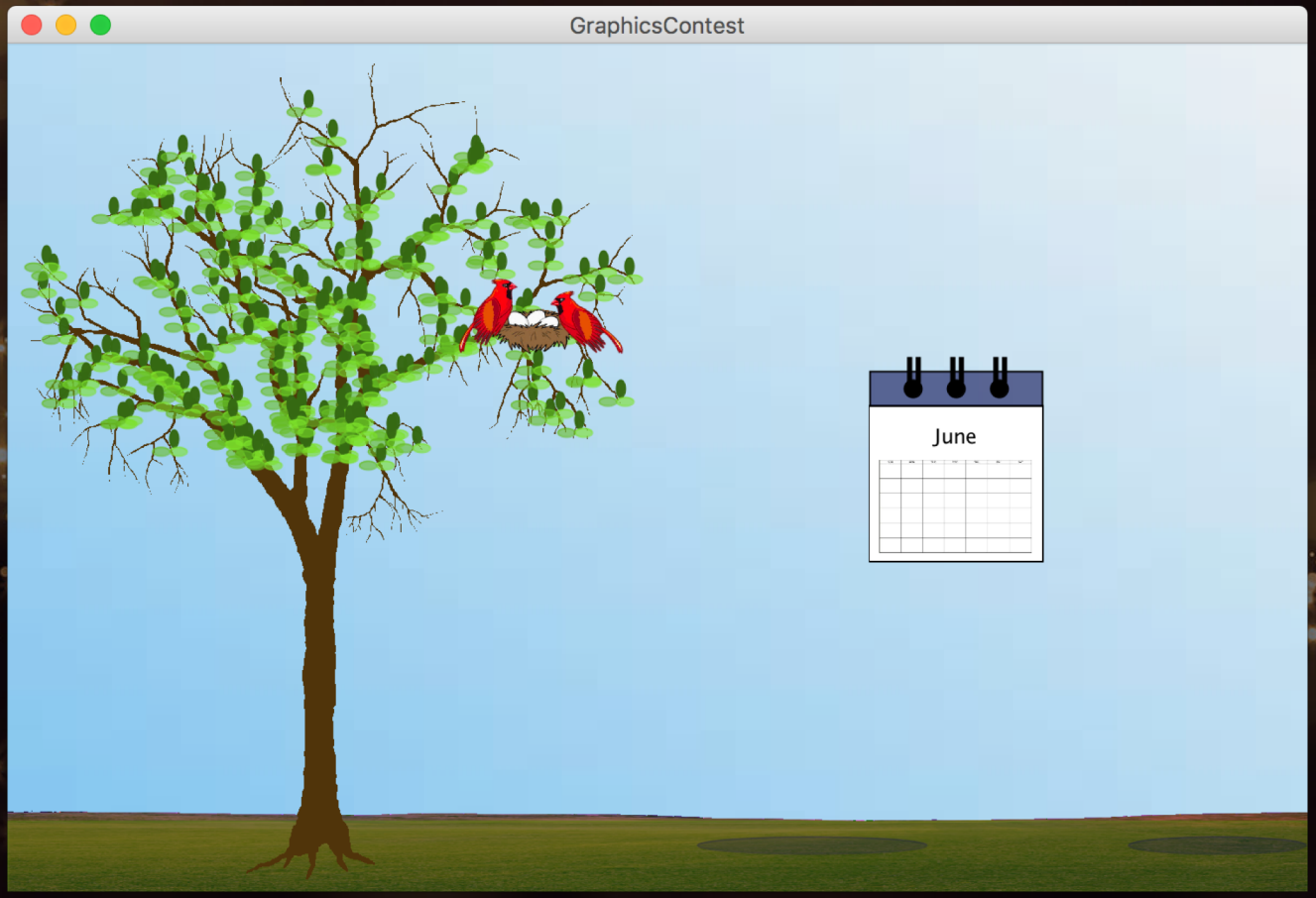

Kills: 26


What should Chris bring to lecture today?
ROMANESCO



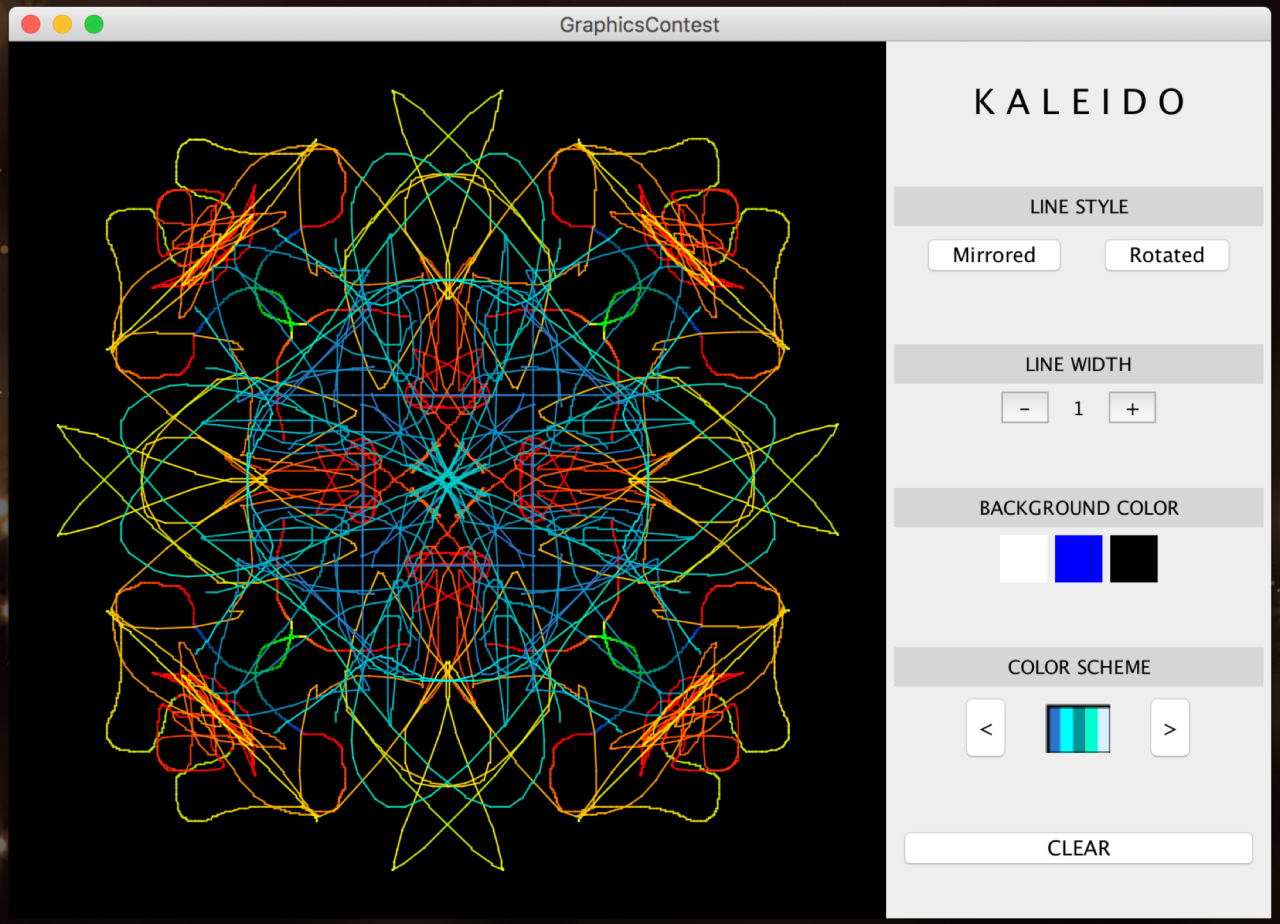


Winner: Aesthetics



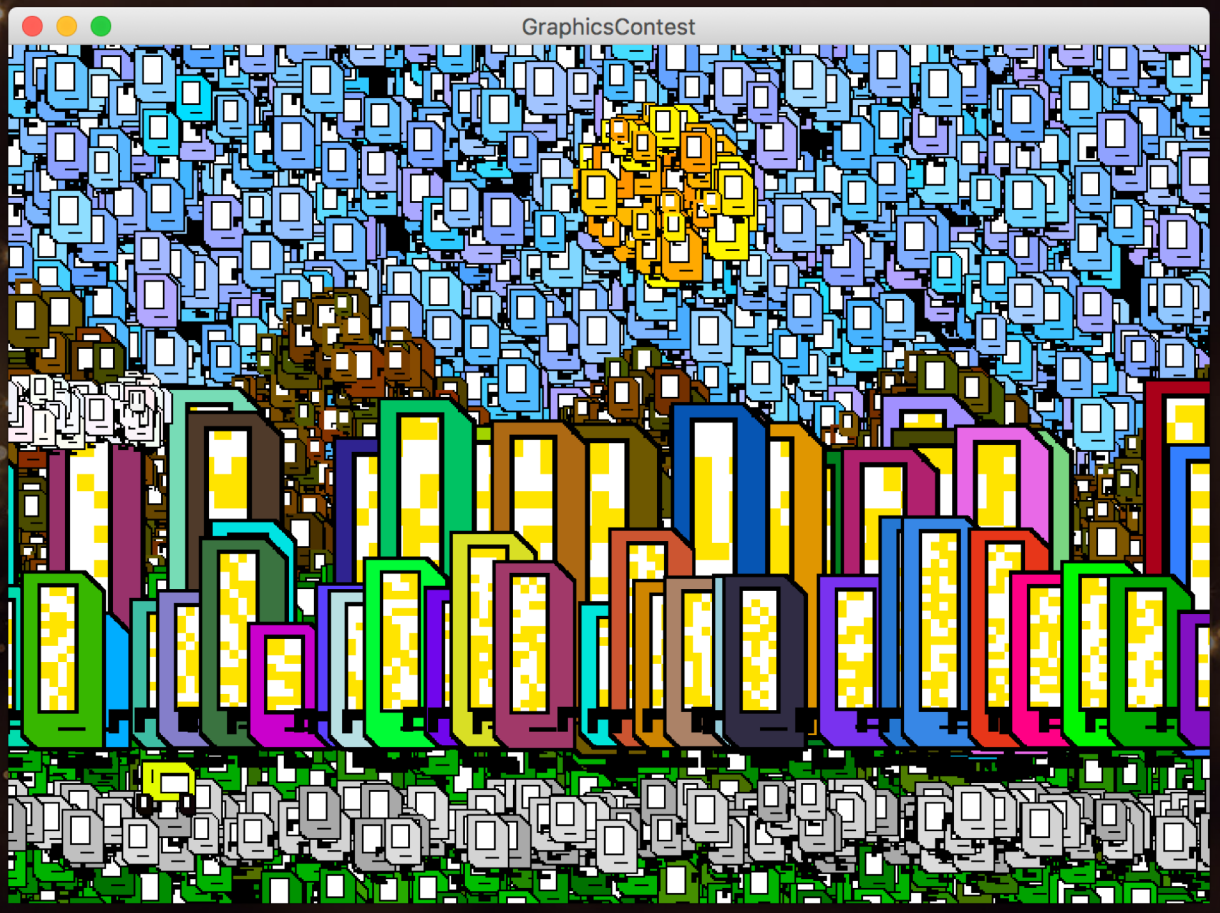
Elina Thadhani

Winner: Algorithmics



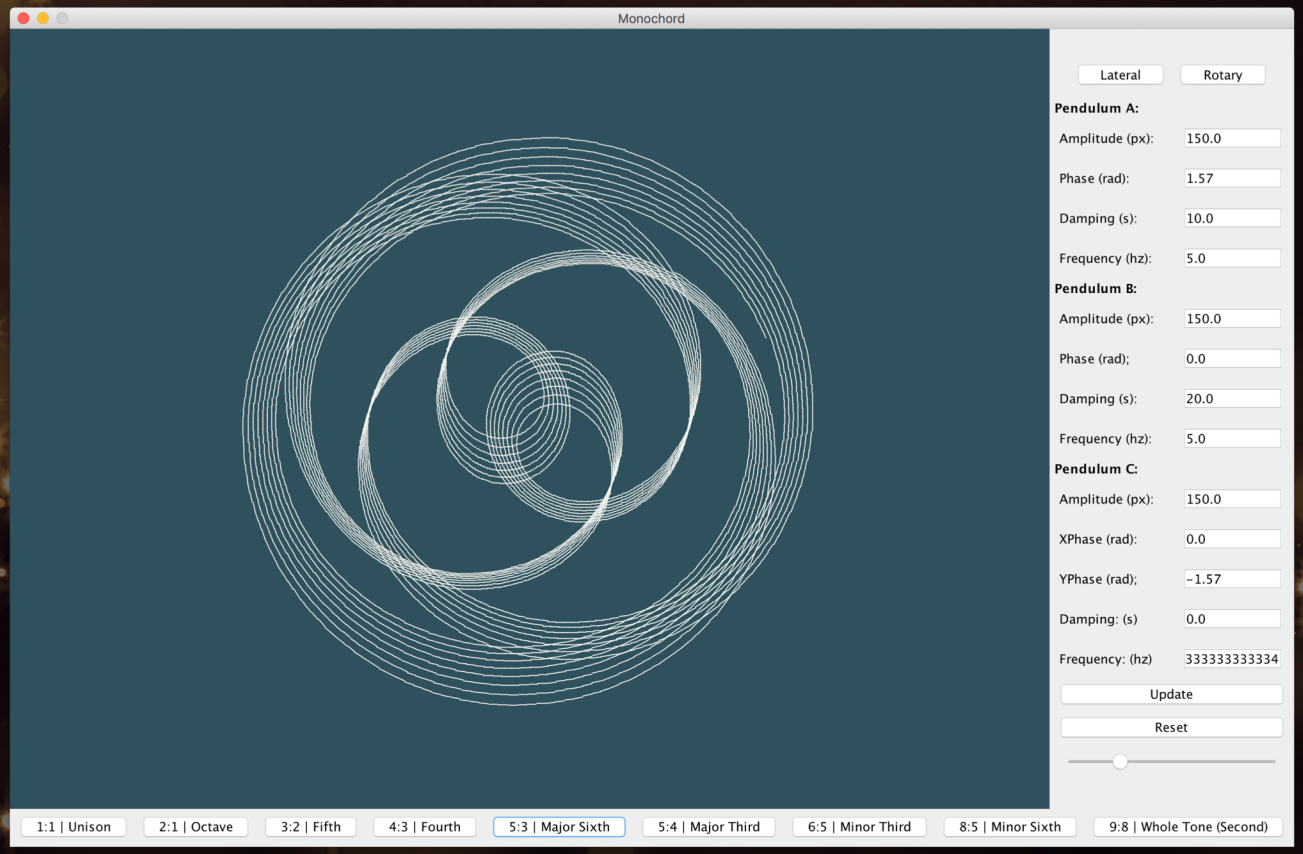
Julianne Crawford

Winner: People's Choice



Nicolas Guillen Barrail

Runner-Up



Nic Becker

Plan for today

- Announcements
- HashMaps
- Classes
- Interactors
- Final Exam Tips

Review: HashMaps

	String	Array	2D Array	ArrayList	HashMap
Model	Sequence of letters or symbols	Fixed length elements in a list	Grid / Matrix of elements	Growable list of elements	Key/Value mapping
Type of element	chars	Objects & Primitives	Objects & Primitives	Objects	Object/Object
Access Elements	str.charAt(i);	arr[i];	arr[r][c];	list.get(i); list.set(i, elem) list.add(elem)	map.put(key, value) map.get(key);
Special notes	Immutable	Watch bounds!	Row, col structure	Just fantastic	Each key must be unique. Unordered
Examples	”Hello world”	Histogram	ImageShop pixels	Hangman words, entries in namesurfer	NSDatabase, FPDatabase

Review: HashMaps

	String	Array	2D Array	ArrayList	HashMap
Model	Sequence of letters or symbols	Fixed length elements in a list	Grid / Matrix of elements	Growable list of elements	Key/Value mapping
Type of element	chars	Objects & Primitives	Objects & Primitives	Objects	Object/Object
Access Elements	str.charAt(i);	arr[i];	arr[r][c];	list.get(i); list.set(i, elem) list.add(elem)	map.put(key, value) map.get(key);
Special notes	Immutable	Watch bounds!	Row, col structure	Just fantastic	Each key must be unique. Unordered
Examples	”Hello world”	Histogram	ImageShop pixels	Hangman words, entries in namesurfer	NSDatabase

Review: HashMaps

- A variable type that represents a collection of **key-value pairs**
- You access values by *key,* and all keys are unique
- Keys and values can be any type of **object** (use wrapper classes to store primitives)
- Resizable – can add and remove pairs
- Has a variety of methods you can use, including *.containsKey*, *.put*, *.get*, etc.

Key Idea: Association

- **Phone book:** name -> phone number
- **Search engine:** URL -> webpage
- **Dictionary**: word -> definition
- **Bank**: account # -> balance
- **Social Network**: name -> profile
- **Counter**: text -> # occurrences
- And many more…

Review: HashMap Operations

- *m*.put(*key*, *value*); Adds a key/value pair to the map.

```
m.put("Eric", "650-123-4567");
```

  - Replaces any previous value for that key.

- *m*.get(*key*) Returns the value paired with the given key.

```
String phoneNum = m.get("Jenny");  // "867-5309"
```

  - Returns null if the key is not found.

- **m.remove**(*key*); Removes the given key and its paired value.

```
m.remove("Rishi");
```

  - Has no effect if the key is not in the map.

key	value
"Jenny"	→ "867-5309"
"Mehran"	→ "123-4567"
"Marty"	→ "685-2181"
"Chris"	→ "947-2176"

Review: HashMap Operations

- *m*.containsKey(*key*); Returns true if the key is in the map, false otherwise

- *m*.size(); Returns the number of key/value pairs in the map.

- To iterate over a map:

```
for (KeyType key : map.keySet()) {
    ValueType value = map.get(key);
    // Do something with key and/or value
}
```

What data structure should I use?

- Use an **array** if…
  - Order matters for your information
  - You know how many elements you will store
  - You need the most efficiency
- Use an **ArrayList** if…
  - Order matters for your information
  - You do not know how many elements you will store, or need to resize
  - You need to use ArrayList methods
- Use a **HashMap** if…
  - Order doesn't matter for your information
  - You need to store an *association* between two types of information
  - You do not know how many elements you will store, or need to resize
  - You need to use HashMap methods

Practice: Anagrams

- Write a program to find all **anagrams** of a word the user types.

```
Type a word [Enter to quit]: scared
Anagrams of scared:
cadres cedars sacred scared
```

- Assume you are given a **`dictionary.txt`** file containing words in the dictionary.

- How can a HashMap help us solve this problem?

Key Idea: Anagrams

- Every word has a *sorted form* where its letters are arranged into alphabetical order.

  `"fare"` → `"aefr"`

  `"fear"` → `"aefr"`

  `"swell"` → `"ellsw"`

  `"wells"` → `"ellsw"`

- Notice that anagrams have the same **sorted form** as each other.
- Assume we provide a **`String sortLetters(String s)`** method that takes a string and returns the string with its characters alphabetically ordered.

Anagrams Solution

```
public void run() {
        HashMap<String, ArrayList<String>> anagrams =
                createAnagramsMap();

        // prompt user for words and look up anagrams in map
        while (true) {
            String word = readLine("Type a word [Enter to quit]: ");
            if (word.length() == 0) {
                break;
            }

            String sorted = sortLetters(word.toLowerCase());
            if (anagrams.containsKey(sorted)) {
                println("Anagrams of " + word + ":");
                println(anagrams.get(sorted));
            } else {
                println("No anagrams for " + word + ".");
            }
        }
}
```

Anagrams Solution

```
public void run() {
        HashMap<String, ArrayList<String>> anagrams =
                createAnagramsMap();

        // prompt user for words and look up anagrams in map
        while (true) {
            String word = readLine("Type a word [Enter to quit]: ");
            if (word.length() == 0) {
                break;
            }

            String sorted = sortLetters(word.toLowerCase());
            if (anagrams.containsKey(sorted)) {
                println("Anagrams of " + word + ":");
                println(anagrams.get(sorted));
            } else {
                println("No anagrams for " + word + ".");
            }
        }
}
```

Anagrams Solution

```
public void run() {
        HashMap<String, ArrayList<String>> anagrams =
                createAnagramsMap();

        // prompt user for words and look up anagrams in map
        while (true) {
            String word = readLine("Type a word [Enter to quit]: ");
            if (word.length() == 0) {
                break;
            }

            String sorted = sortLetters(word.toLowerCase());
            if (anagrams.containsKey(sorted)) {
                println("Anagrams of " + word + ":");
                println(anagrams.get(sorted));
            } else {
                println("No anagrams for " + word + ".");
            }
        }
}
```

Anagrams Solution

```
public void run() {
        HashMap<String, ArrayList<String>> anagrams =
                createAnagramsMap();

        // prompt user for words and look up anagrams in map
        while (true) {
            String word = readLine("Type a word [Enter to quit]: ");
            if (word.length() == 0) {
                break;
            }

            String sorted = sortLetters(word.toLowerCase());
            if (anagrams.containsKey(sorted)) {
                println("Anagrams of " + word + ":");
                println(anagrams.get(sorted));
            } else {
                println("No anagrams for " + word + ".");
            }
        }
}
```

Anagrams Solution

```
public void run() {
        HashMap<String, ArrayList<String>> anagrams =
                createAnagramsMap();

        // prompt user for words and look up anagrams in map
        while (true) {
            String word = readLine("Type a word [Enter to quit]: ");
            if (word.length() == 0) {
                break;
            }

            String sorted = sortLetters(word.toLowerCase());
            if (anagrams.containsKey(sorted)) {
                println("Anagrams of " + word + ":");
                println(anagrams.get(sorted));
            } else {
                println("No anagrams for " + word + ".");
            }
        }
}
```

Anagrams Solution

```
public void run() {
        HashMap<String, ArrayList<String>> anagrams =
                createAnagramsMap();

        // prompt user for words and look up anagrams in map
        while (true) {
            String word = readLine("Type a word [Enter to quit]: ");
            if (word.length() == 0) {
                break;
            }

            String sorted = sortLetters(word.toLowerCase());
            if (anagrams.containsKey(sorted)) {
                println("Anagrams of " + word + ":");
                println(anagrams.get(sorted));
            } else {
                println("No anagrams for " + word + ".");
            }
        }
}
```

Anagrams Solution, Part 2

```
/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String,
            ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
```

Anagrams Solution, Part 2

```
/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String,
            ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
```

Anagrams Solution, Part 2

```
/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String,
            ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
```

Anagrams Solution, Part 2

```
/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String,
            ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
```

Anagrams Solution, Part 2

```
/*
 * This method reads in the dictionary.txt file and creates a map
 * from a sorted string of characters to all words made up of those
 * characters. e.g. “acers” -> {“scare”, “cares”,...}
 */
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<String,
            ArrayList<String>>();

    try {
        Scanner scanner = new Scanner(new File("dictionary.txt"));
        while (scanner.hasNextLine()) {
            String word = scanner.nextLine();
            String sorted = sortLetters(word);
            ...
```

Anagrams Solution, Part 2

```
            /* Either get the current word list,
             * or make a new empty one. */
            ArrayList<String> words;
            if (anagrams.containsKey(sorted)) {
                words = anagrams.get(sorted);
            } else {
                words = new ArrayList<String>();
            }

            // Update the list with our new word
            words.add(word);
            anagrams.put(sorted, words);
        }
        scanner.close();
    } catch (IOException ex) {
        println("Error reading file dictionary.txt.");
    }

    return anagrams;
}
```

Anagrams Solution, Part 2

```
            /* Either get the current word list,
             * or make a new empty one. */
            ArrayList<String> words;
            if (anagrams.containsKey(sorted)) {
                words = anagrams.get(sorted);
            } else {
                words = new ArrayList<String>();
            }

            // Update the list with our new word
            words.add(word);
            anagrams.put(sorted, words);
        }
        scanner.close();
    } catch (IOException ex) {
        println("Error reading file dictionary.txt");
    }

    return anagrams;
}
```

Anagrams Solution, Part 2

```
            /* Either get the current word list,
             * or make a new empty one. */
            ArrayList<String> words;
            if (anagrams.containsKey(sorted)) {
                words = anagrams.get(sorted);
            } else {
                words = new ArrayList<String>();
            }

            // Update the list with our new word
            words.add(word);
            anagrams.put(sorted, words);
        }
        scanner.close();
    } catch (IOException ex) {
        println("Error reading file dictionary.txt");
    }

    return anagrams;
}
```

Anagrams Solution, Part 2

```
            /* Either get the current word list,
             * or make a new empty one. */
            ArrayList<String> words;
            if (anagrams.containsKey(sorted)) {
                words = anagrams.get(sorted);
            } else {
                words = new ArrayList<String>();
            }

            // Update the list with our new word
            words.add(word);
            anagrams.put(sorted, words);
        }
        scanner.close();
    } catch (IOException ex) {
        println("Error reading file dictionary.txt");
    }

    return anagrams;
}
```

Anagrams Solution, Part 2

```
            /* Either get the current word list,
             * or make a new empty one. */
            ArrayList<String> words;
            if (anagrams.containsKey(sorted)) {
                words = anagrams.get(sorted);
            } else {
                words = new ArrayList<String>();
            }

            // Update the list with our new word
            words.add(word);
            anagrams.put(sorted, words);
        }
        scanner.close();
    } catch (IOException ex) {
        println("Error reading file dictionary.txt.");
    }

    return anagrams;
}
```

Plan for today

- Announcements
- HashMaps
- Classes
- Interactors
- Final Exam Tips

What Is A Class?

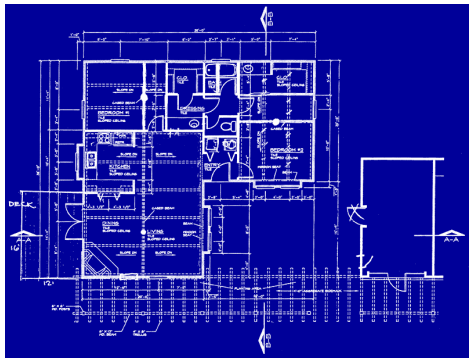
A class defines a new variable type.

Classes Are Like Blueprints

iPod blueprint (class)

state:
- current song
- volume
- battery life

behavior:
- power on/off
- change station/song
- change volume
- choose random song



constructs

iPod (variable) #1

state:
- song = "1,000,000 Miles"
- volume = 17
- battery life = 2.5 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

iPod (variable) #2

state:
- song = "Letting You"
- volume = 9
- battery life = 3.41 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

iPod (variable) #3

state:
- song = "Discipline"
- volume = 24
- battery life = 1.8 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

Creating A New Class

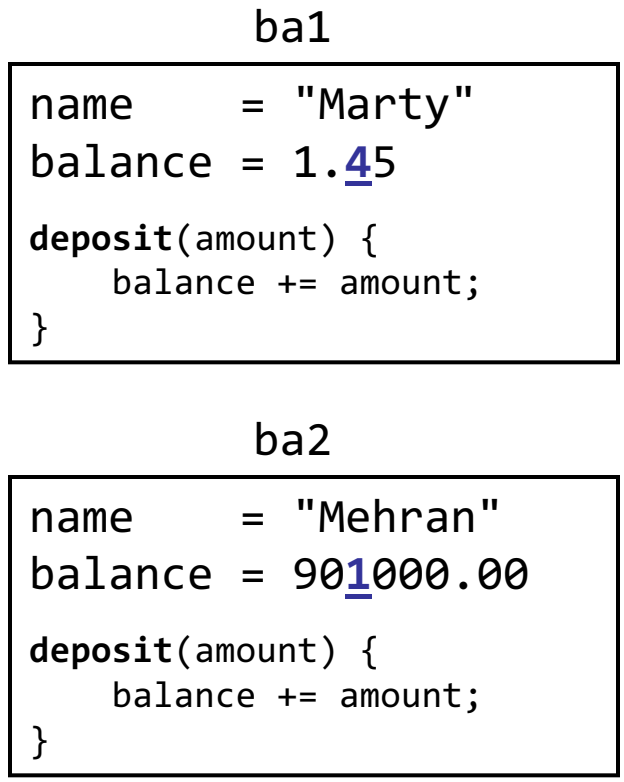
1. **What information is inside this new variable type?** These are its instance variables.
2. **What can this new variable type do?** These are its public methods.
3. **How do you create a variable of this type?** This is the constructor.

Defining Methods In Classes

Methods defined in classes can be called **on an instance of that class**.

When one of these methods executes, it can reference **that object's copy** of instance variables.

```
ba1.deposit(0.20);
ba2.deposit(1000.00);
```



This means calling one of these methods on different objects has *different effects.*

Constructors

```
BankAccount ba1 = new BankAccount();

BankAccount ba2 = new BankAccount("Nick", 50);
```

The constructor is executed when a new object is created.

Example: BankAccount

```
public class BankAccount {
  // Step 1: the data inside a BankAccount
  private String name;
  private double balance;

  // Step 2: the things a BankAccount can do (omitted)
  // Step 3: how to create a BankAccount
  public BankAccount(String accountName, double startBalance) {
       name = accountName;
       balance = startBalance;
  }

  public BankAccount(String accountName) {
       name = accountName;
       balance = 0;
  }
}
```

Using Constructors

```
BankAccount ba1 =
    new BankAccount("Marty", 1.25);
```

ba1

```
name    = "Marty"
balance = 1.25

BankAccount(nm, bal) {
    name = nm;
    balance = bal;
}
```

```
BankAccount ba2 =
    new BankAccount("Mehran", 900000.00);
```

ba2

```
name    = "Mehran"
balance = 900000.00

BankAccount(nm, bal) {
    name = nm;
    balance = bal;
}
```

- When you call a constructor (with **new**):
  - Java creates a new object of that class.
  - The constructor runs, on that new object.
  - The newly created object is returned to your program.

Practice: Airplane!

Let's write a class called **Airplane** that implements functionality for boarding/unboarding passengers from a plane.

```
int capacity = readInt("Capacity? ");
Airplane plane = new Airplane(capacity);

// Board passengers
while (!plane.isFull()) {
    String passengerName = readLine("Name: ");
    boolean priority = readBoolean("Priority? (true/false) ");
    plane.boardPassenger(passengerName, priority);
}

// fly...

// Unboard passengers
while (!plane.isEmpty()) {
    String passengerName = plane.unboardPassenger();
    println("Unboarded " + passengerName);
}
```

Practice: Airplane!

Let's write a class called **Airplane** that implements the following functionality for boarding/unboarding passengers from a plane.

```
// Creates a new airplane with the given capacity
public Airplane(int capacity);

/* Boards 1 passenger, at front if they are priority, or
 * back otherwise */
public void boardPassenger(String name, boolean priority);

public boolean isFull();
public boolean isEmpty();

/* Unboards and returns next passenger, or null if there
 * are no more passengers. */
public String unboardPassenger();
```

Creating A New Class

1. **What information is inside this new variable type?** These are its instance variables.
2. **What can this new variable type do?** These are its public methods.
3. **How do you create a variable of this type?** This is the constructor.

Practice: Airplane!

```
public class Airplane {
    private ArrayList<String> passengers;
    private int capacity;

    ...
```

Creating A New Class

1. **What information is inside this new variable type?** These are its instance variables.
2. **What can this new variable type do?** These are its public methods.
3. **How do you create a variable of this type?** This is the constructor.

boardPassenger

```
...
public void boardPassenger(String name, boolean priority) {
        if (!isFull()) {
                if (priority) {
                        passengers.add(0, name);
                } else {
                        passengers.add(name);
                }
        }
}
...
```

isFull

```
...
public boolean isFull() {
      return capacity == passengers.size();
}
...
```

isEmpty

```
...
public boolean isEmpty() {
      return passengers.isEmpty();
}
...
```

unboardPassenger

```
...
public String unboardPassenger() {
      if (!isEmpty()) {
            return passengers.remove(0);
      }
      return null;
}
...
```

Creating A New Class

1. **What information is inside this new variable type?** These are its instance variables.
2. **What can this new variable type do?** These are its public methods.
3. **How do you create a variable of this type?** This is the constructor.

Practice: Airplane!

```
// Private instance variables
private ArrayList<String> passengers;
private int capacity;

// Constructor
public Airplane(int numSeats) {
      capacity = numSeats;
      passengers = new ArrayList<String>();
}
...
```

Plan for today

- Announcements
- HashMaps
- Classes
- Interactors
- Final Exam Tips

Review: Interactors

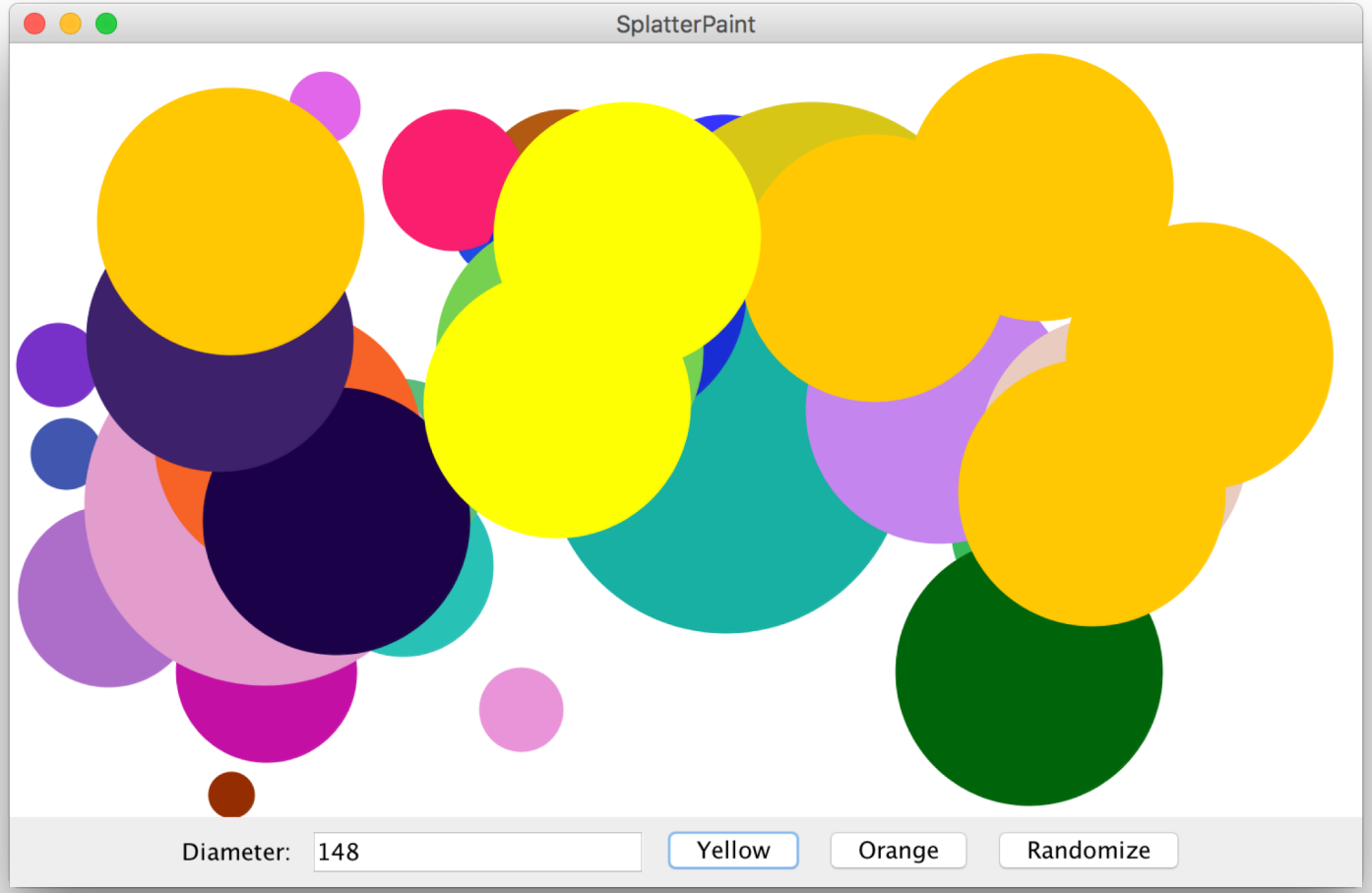
1. Add interactors in **`init()`**
2. **`addActionListeners()`** to listen for button presses
3. **`.addActionListener(this)`** on text fields for ENTER
    1. Plus (usually) **`setActionCommand(command)`**
4. Implement **`actionPerformed`**
5. Java will call **`actionPerformed`** whenever an action event occurs.

Interactors

```
public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("My Interactor")) {
                ...
        }
}

// ... equivalent to ...
public void actionPerformed(ActionEvent e) {
        if (e.getSource() == myInteractor) {
                ...
        }
}
```

Practice: SplatterPaint



Practice: SplatterPaint

- Add a new ”splatter” (e.g. GOval) every time the user clicks on the “Yellow” or “Orange” buttons.
- If the user hits “ENTER” instead, add a Yellow splatter.
- The splatter’s diameter should be what is entered in the text box (assume it’s valid), and placed randomly **entirely onscreen.**
- If the user clicks “Randomize”, **all onscreen splatters** should be changed to different random colors.

Solution: SplatterPaint

- If “Yellow” is clicked OR IF ENTER IS PRESSED, add a splatter with the diameter in the text field to the screen in **yellow**
- If “Orange” is clicked, add a splatter with the diameter in the text field to the screen in **orange**
- If “Randomize” is clicked, iterate over all splatters and change their color

SplatterPaint.java

```
public class SplatterPaint extends GraphicsProgram {
	// The width of the diameter text field (in chars)
	private static final int DIAMETER_FIELD_WIDTH = 15;

	// The field for entering the splatter diameter
	private JTextField diameterField;

	// A list of all onscreen splatters
	private ArrayList<GOval> splatters;

...

}
```

SplatterPaint.java

```
public void init() {
        splatters = new ArrayList<GOval>();

        // Add the text field
        add(new JLabel("Diameter: "), SOUTH);
        diameterField = new JTextField(DIAMETER_FIELD_WIDTH);
        diameterField.setActionCommand("Yellow");
        diameterField.addActionListener(this);
        add(diameterField, SOUTH);
        // Add the buttons
        add(new JButton("Yellow"), SOUTH);
        add(new JButton("Orange"), SOUTH);
        add(new JButton("Randomize"), SOUTH);
        addActionListeners();
}
```

SplatterPaint.java

```
public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Yellow")) {
                // Add a yellow circle
                int diameter =
                        Integer.parseInt(diameterField.getText());
                addSplatter(diameter, Color.yellow);
        } else if (e.getActionCommand().equals("Orange")) {
                // Add an orange circle
                int diameter =
                        Integer.parseInt(diameterField.getText());
                addSplatter(diameter, Color.orange);
        } else if (e.getActionCommand().equals("Randomize")) {
                // Randomize all existing splatters
                randomizeColors();
        }
}
```

SplatterPaint.java

```
private void addSplatter(int diameter, Color color) {
	int x = RandomGenerator.getInstance()
		.nextInt(0, getWidth() - diameter);
	int y = RandomGenerator.getInstance()
		.nextInt(0, getHeight() - diameter);

	GOval splatter = new GOval(x, y, diameter, diameter);
	splatter.setFilled(true);
	splatter.setColor(color);
	add(splatter);
	splatters.add(splatter);
}
```

SplatterPaint.java

```
private void randomColors() {
	for (GOval splatter : splatters) {
		Color newColor =
			RandomGenerator.getInstance().nextColor();
		splatter.setColor(newColor);
	}
}
```

Recap

- Announcements
- HashMaps
- Classes
- Interactors
- Final Exam Tips

Final Exam Tips

- Look over all problems before starting
- Pseudocode!
- Show your work
- Practice practice practice
- Review common programming paradigms
  - Iterating over a string
  - actionPerformed
  - …

Final Exam Tips

- Reflect on how much you've learned in just 1 quarter!