CS 106A, Lecture 28
Final Exam Review 2
Plan for today

• Announcements
• HashMaps
• Classes
• Inheritance
• Interactors
• Closing Remarks
Plan for today

• Announcements
• HashMaps
• Classes
• Inheritance
• Interactors
• Closing Remarks
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.
HashMap Examples

- **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.
  
  ```java
  m.put("Eric", "650-123-4567");
  ```
  - Replaces any previous value for that key.

- `m.get(key)` Returns the value paired with the given key.
  
  ```java
  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.
  
  ```java
  m.remove("Rishi");
  ```
  - Has no effect if the key is not in the map.

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Jenny&quot;</td>
<td>&quot;867-5309&quot;</td>
</tr>
<tr>
<td>&quot;Mehran&quot;</td>
<td>&quot;123-4567&quot;</td>
</tr>
<tr>
<td>&quot;Marty&quot;</td>
<td>&quot;685-2181&quot;</td>
</tr>
<tr>
<td>&quot;Chris&quot;</td>
<td>&quot;947-2176&quot;</td>
</tr>
</tbody>
</table>
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:

```java
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 the following:
  - A `dictionary.txt` file containing words in the dictionary
  - A method `private String sortLetters(String s)` method that takes a string and returns the string with its characters alphabetically ordered.

- 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.
public void run() {
    HashMap<String, ArrayList<String>> anagrams =
        createAnagramsMap();
    // prompt user for words and look up anagrams in map
    String word = readLine("Type a word [Enter to quit]: ");
    while (word.length() > 0) {
        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
        word = readLine("Type a word [Enter to quit]: ");
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams =
        createAnagramsMap();

    // prompt user for words and look up anagrams in map
    String word = readLine("Type a word [Enter to quit]: ");
    while (word.length() > 0) {
        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
        word = readLine("Type a word [Enter to quit]: ");
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams =
        createAnagramsMap();
    // prompt user for words and look up anagrams in map
    String word = readLine("Type a word [Enter to quit]: ");
    while (word.length() > 0) {
        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
        word = readLine("Type a word [Enter to quit]: ");
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams = createAnagramsMap();
    // prompt user for words and look up anagrams in map
    String word = readLine("Type a word [Enter to quit]: ");
    while (word.length() > 0) {
        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
        word = readLine("Type a word [Enter to quit]: ");
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams =
        createAnagramsMap();
    // prompt user for words and look up anagrams in map
    String word = readLine("Type a word [Enter to quit]: ");
    while (word.length() > 0) {
        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for " + word + ".");
        }
        word = readLine("Type a word [Enter to quit]: ");
    }
}
public void run() {
    HashMap<String, ArrayList<String>> anagrams =
        createAnagramsMap();
    // prompt user for words and look up anagrams in map
    String word = readLine("Type a word [Enter to quit]: ");
    while (word.length() > 0) {
        String sorted = sortLetters(word.toLowerCase());
        if (anagrams.containsKey(sorted)) {
            println("Anagrams of " + word + ":");
            println(anagrams.get(sorted));
        } else {
            println("No anagrams for ", word, ".");
        }
        word = readLine("Type a word [Enter to quit]: ");
    }
}
// Returns a new map from a sorted word to all words created
// from those letters - e.g. “acers” -> {“scare”, “cares”,...}
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams =
        new HashMap<>();

    try {
        Scanner scanner =
            new Scanner(new File("res/dictionary.txt"));

        while (scanner.hasNext()) {
            String word = scanner.next();
            String sorted = sortLetters(word);
            ...
        }
    }
}
// Returns a new map from a sorted word to all words created
// from those letters - e.g. “acers” -> {“scare”, “cares”,...}
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams =
        new HashMap<>();
    try {
        Scanner scanner =
            new Scanner(new File("res/dictionary.txt"));
        while (scanner.hasNext()) {
            String word = scanner.next();
            String sorted = sortLetters(word);
            ...
        }
    } catch (Exception e) {
        // handle exception...
    }
}
// Returns a new map from a sorted word to all words created
// from those letters - e.g. “acers” -> {“scare”, “cares”,...}
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<>();

    try {
        Scanner scanner =
            new Scanner(new File("res/dictionary.txt"));
        while (scanner.hasNext()) {
            String word = scanner.next();
            String sorted = sortLetters(word);
            ...
        }
    }
}
// Returns a new map from a sorted word to all words created // from those letters - e.g. “acers” -> {“scare”, “cares”,...} private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams = new HashMap<>();
    try {
        Scanner scanner =
            new Scanner(new File("res/dictionary.txt"));
        while (scanner.hasNext()) {
            String word = scanner.next();
            String sorted = sortLetters(word);
            ...
        }
    }
}
// Returns a new map from a sorted word to all words created
// from those letters - e.g. “acers” -> {“scare”, “cares”,...}
private HashMap<String, ArrayList<String>> createAnagramsMap() {
    HashMap<String, ArrayList<String>> anagrams =
    new HashMap<>();
    try {
        Scanner scanner =
            new Scanner(new File("res/dictionary.txt"));
        while (scanner.hasNext()) {
            String word = scanner.next();
            String sorted = sortLetters(word);
            ...
        }
    }
}
Anagrams Solution, Part 2

```java
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<>();
}

words.add(word);
anagrams.put(sorted, words);
}
scanner.close();
} catch (IOException ex) {
    println("Error reading file.");
}
return anagrams;
```
```java
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<>();
}
words.add(word);
anagrams.put(sorted, words);
}
scanner.close();
} catch (IOException ex) {
    println("Error reading file.");
}
return anagrams;
}```
Anagrams Solution, Part 2

```java
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<>();
}

words.add(word);
anagrams.put(sorted, words);
```

```java
scanner.close();
}
```
Anagrams Solution, Part 2

```java
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<>();
}

words.add(word);
anagrams.put(sorted, words);

scanner.close();
}
```

```java
} catch (IOException ex) {
    println("Error reading file.");
}
return anagrams;
```
Anagrams Solution, Part 2

```java
ArrayList<String> words;
if (anagrams.containsKey(sorted)) {
    words = anagrams.get(sorted);
} else {
    words = new ArrayList<>();
}

words.add(word);
anagrams.put(sorted, words);
}
scanner.close();

} catch (IOException ex) {
    println("Error reading file.");
}
return anagrams;
```
Plan for today

• Announcements
• HashMaps
• Classes
• Inheritance
• Interactors
• Closing Remarks
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

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.
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.

```java
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.
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);

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

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

```java
Airplane plane = new Airplane(100);
while (!plane.isFull()) {
    String passengerName = readLine("Name: ");
    boolean priority = readBoolean("Priority? ");
    plane.boardPassenger(passengerName, priority);
}
// fly...
while (!plane.isEmpty()) {
    String passengerName = plane.unboardPassenger();
    println("Unboarded " + passengerName);
}
```
Let’s write a class called **Airplane** that implements functionality for boarding/unboarding passengers from a plane.

It should implement the following methods:

```java
/**
 * Constructor should take 1 parameter (plane capacity)
 */

/**
 * Boards 1 passenger, at front or back **/
 * @param name The passenger's name
 * @param priority If true, passenger has priority
 */
public void boardPassenger(String name, boolean priority);

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

/**
 * Unboards and returns next passenger **/
 * @return The passenger who just got off
 */
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.
public class Airplane {
    private ArrayList<String> passengers;
    private int capacity;

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

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

...
public boolean isEmpty() {
    return passengers.isEmpty();
}
... 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.
// Private instance variables
private ArrayList<String> passengers;
private int capacity;

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

Plan for today

• Announcements
• HashMaps
• Classes
• Inheritance
• Interactors
• Closing Remarks
Inheritance lets us relate our variable types to one another.
Variable types can seem to “inherit” from each other. We don’t want to have to duplicate code for each one!
Using Inheritance

public class Name extends Superclass {

    // Example:
    public class Programmer extends Employee {
        ...
    }

    • By extending Employee, this tells Java that Programmer can do everything an Employee can do, plus more.
    • Programmer automatically inherits all of the code from Employee!
    • The superclass is Employee, the subclass is Programmer.
public class Programmer extends Employee {
    private int timeCoding;
    ...
    public void code() {
        timeCoding += 10;
    }
}

Programmer rishi = new Programmer("Rishi");
rishi.code(); // from Programmer
rishi.promote(); // from Employee!
Example: KarelProgrammer

```java
public class KarelProgrammer extends Programmer {
    private int numBeepersPicked;
    ...
    public void pickBeepers() {
        numBeepersPicked += 2;
    }
}

...  
KarelProgrammer nick = new KarelProgrammer("Nick");
nick.pickBeepers(); // from KarelProgrammer
nick.code();         // from Programmer!
nick.promote();     // From Employee!
```
Extending GCanvas

• Sometimes, we want to be able to have all of our graphics-related code in a separate file.

• To do this, instead of using the provided `GraphicsProgram` canvas, we define our own subclass of `GCanvas`, have our program extend `Program`, and add our own canvas ourselves.

• Then, all graphics-related code can go in our `GCanvas` subclass.
Extending GCanvas

```java
public class MyCanvas extends GCanvas {
    public void addCenteredSquare(int size) {
        GRect rect = new GRect(size, size);
        int x = getWidth() / 2.0 - rect.getWidth() / 2.0;
        int y = getHeight() / 2.0 - rect.getHeight() / 2.0;
        add(rect, x, y);
    }
}
```
public class Graphics extends Program {
    public void run() {
        // We have to make our own GCanvas now
        MyCanvas canvas = new MyCanvas();
        add(canvas);

        canvas.addCenteredSquare(20);
    }
}
Plan for today

• Announcements
• HashMaps
• Classes
• Inheritance
• Interactors
• Closing Remarks
1. Add interactors in \texttt{init()}

2. \texttt{addActionListeners()} to listen for button presses

3. \texttt{.addActionListener(this)} on text fields for ENTER
   
   1. Plus (usually) \texttt{setActionCommand(command)}

4. Implement \texttt{actionPerformed}

5. Java will call \texttt{actionPerformed} whenever an action event occurs.
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.

How would we implement this? How should we design our custom GCanvas class?
Solution: SplatterPaint

• Have a `SplatterPaint` class that extends `Program`, to which we add our custom `GCanvas`.

• Have a `SplatterPaintCanvas` class that extends `GCanvas`, which contains all graphics logic. It could have public methods such as:
  - `public void addSplatter(int diameter, Color color)`
    Adds a splatter with the given diameter and color randomly onscreen
  
  - `public void randomColors()` Randomizes the colors of all onscreen splatters.
public class SplatterPaint extends Program {
   // 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;

   // The custom canvas that displays all splatters
   private SplatterPaintCanvas canvas;

   ...
}

public void init() {
    canvas = new SplatterPaintCanvas();
    add(canvas);
    // 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();
}
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("Yellow")) {
        // Add a yellow circle
        int diameter =
            Integer.parseInt(diameterField.getText());
        canvas.addSplatter(diameter, Color.yellow);
    } else if (e.getActionCommand().equals("Orange")) {
        // Add an orange circle
        int diameter =
            Integer.parseInt(diameterField.getText());
        canvas.addSplatter(diameter, Color.orange);
    } else if (e.getActionCommand().equals("Randomize")) {
        // Randomize all existing splatters
        canvas.randomColors();
    }
}
public class SplatterPaintCanvas extends GCanvas {
    ArrayList<GOval> splatters;

    public SplatterPaintCanvas() {
        splatters = new ArrayList<>();
    }

    ...

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

• Announcements
• HashMaps
• Classes
• Inheritance
• Interactors
• Closing Remarks
Generalization

Before

After
Variables Are Like Boxes!
Methods Are Like Toasters!

parameter(s)  return
A Variable Love Story
A Variable Love Story

origin
Matrices
Anyone can be a computer scientist.
Computer science is using computing to solve problems.
Anyone is welcome to be a computer scientist.
We hope 106A excited you about learning more computer science!