CS106A
Midterm 2 Review

Miles Seiver
1p - 3p
2 March 2014

The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings

The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings

Midterm logistics

- Midterm is this **Wednesday** from **7pm - 10pm**
- Location is based on your **last name**:
  - **Abr - Che:** Gates B01
  - **Chi - Erd:** Gates B03
  - **Esp - Fre:** Gates B12
  - **Fu - Kea:** SkillAud
  - **Kel - Lim:** HerrinT175
  - **Lin - Oul:** Hewlett201
  - **Pad - Ren:** 200-205
  - **Rey - San:** 380-380W
  - **Sar - Sta:** 380-380X
  - **Ste - Tse:** 380-380Y
  - **Tsk - Zhu:** 420-041

Exam is open book, open notes, closed computer

The examination is open-book (specifically the course textbook *The Art and Science of Java* and the Karel the Robot coursereader) and you may make use of any handouts, course notes/slides, printouts of your programs or other notes you've taken in the class. You may not, however, use a computer of any kind (i.e., you cannot use laptops on the exam).

Exam is open book, open notes, closed computer

The examination is open-book (specifically the course textbook *The Art and Science of Java* and the Karel the Robot coursereader) and you may make use of any handouts, course notes/slides, printouts of your programs or other notes you've taken in the class. You may not, however, use a computer of any kind (i.e., you cannot use laptops on the exam).

Exam is open book, open notes, closed computer

The examination is open-book (specifically the course textbook *The Art and Science of Java* and the Karel the Robot coursereader) and you may make use of any handouts, course notes/slides, printouts of your programs or other notes you've taken in the class. You may not, however, use a computer of any kind (i.e., you cannot use laptops on the exam).

Coverage

The second midterm exam covers the material presented throughout the class (with the exception of the Karel material). You are responsible for all topics covered in lectures up through and including Wednesday's lecture and for topics from the assignments.

Coverage

The second midterm exam covers the material presented throughout the class (with the exception of the Karel material). You are responsible for all topics covered in lectures up through and including Wednesday's lecture and for topics from the assignments.

Coverage

The second midterm exam covers the material presented throughout the class (with the exception of the Karel material). You are responsible for all topics covered in lectures up through and including Wednesday's lecture and for topics from the assignments.

Key topics

Key topics

- Primitives

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
- constants

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
- constants
- pass by reference vs. by value

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
- constants
- pass by reference vs. by value
- local vs. instance

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
- constants
- pass by reference vs. by value
- local vs. instance
- scope (and masking)
- Methods

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)
- Methods
  - `return` statements

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)
- Methods
  - `return` statements
  - parameters

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)
- Methods
  - `return` statements
  - parameters
- `String`

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)
- Methods
  - `return` statements
  - parameters
- `String`
- Graphics and animation

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)
- Methods
  - `return` statements
  - parameters
- `String`
- Graphics and animation
- Mouse interaction

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while` (`true`), `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)
- Methods
  - `return` statements
  - parameters
- `String`
- Graphics and animation
- Mouse interaction
- `RandomGenerator`

Key topics

- Primitives
  - `int`, `boolean`, `double`, `char`
  - type conversion (typecasting)
- Control structures
  - `if` (`else`), `switch`
  - loops: `for`, `while`, `while (true)`, `break`, `continue`
- Operators
  - (`+`, `%`, `=`, `<=`, `>`, `==`, `&&`, `||`, `!`, etc.)
- Variables
  - constants
  - pass by reference vs. by value
  - local vs. instance
  - scope (and masking)
- Methods
  - `return` statements
  - parameters
- `String`
- Graphics and animation
- Mouse interaction
- `RandomGenerator`

How do you convert a `double` to an `int`?

(typecasting)

```
double x = 5.2349;

println(_________); // 5
```

How do you convert a `double` to an `int`?

(typecasting)

```
double x = 5.2349;

println((int)x);   // 5
```

Key topics continued

Key topics continued

- Classes

Key topics continued

- Classes
  - inheritance: extends, implements

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
  - access (`public` vs. `private`)

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
  - access (`public` vs. `private`)
- Graphs and networks

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays
  - 1D and 2D

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
  - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays
  - 1D and 2D
- `ArrayList`

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays
  - 1D and 2D
- `ArrayList`
- `HashMap`

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays
  - 1D and 2D
- `ArrayList`
- `HashMap`
- Iterators

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays
  - 1D and 2D
- `ArrayList`
- `HashMap`
- Iterators
- Interactors

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays
  - 1D and 2D
- `ArrayList`
- `HashMap`
- Iterators
- Interactors
- Data structure design

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)
- Arrays
  - 1D and 2D
- `ArrayList`
- `HashMap`
- Iterators
- Interactors
- Data structure design
- `GImage`

Key topics continued

- Classes
  - inheritance: extends, implements
  - constructors
    - access (`public` vs. `private`)
- Graphs and networks
- File reading
  - Exception handling (`try-catch`)

- Arrays
  - 1D and 2D
- `ArrayList`
- `HashMap`
- Iterators
- Interactors
- Data structure design
- `GImage`

Other tips

Other tips

- Style doesn’t matter. You don’t need to comment.

Other tips

- Style doesn’t matter. You don’t need to comment.
  - (but…)

Other tips

- Style doesn’t matter. You don’t need to comment.
  - (but…)
- Don’t worry about imports.

Other tips

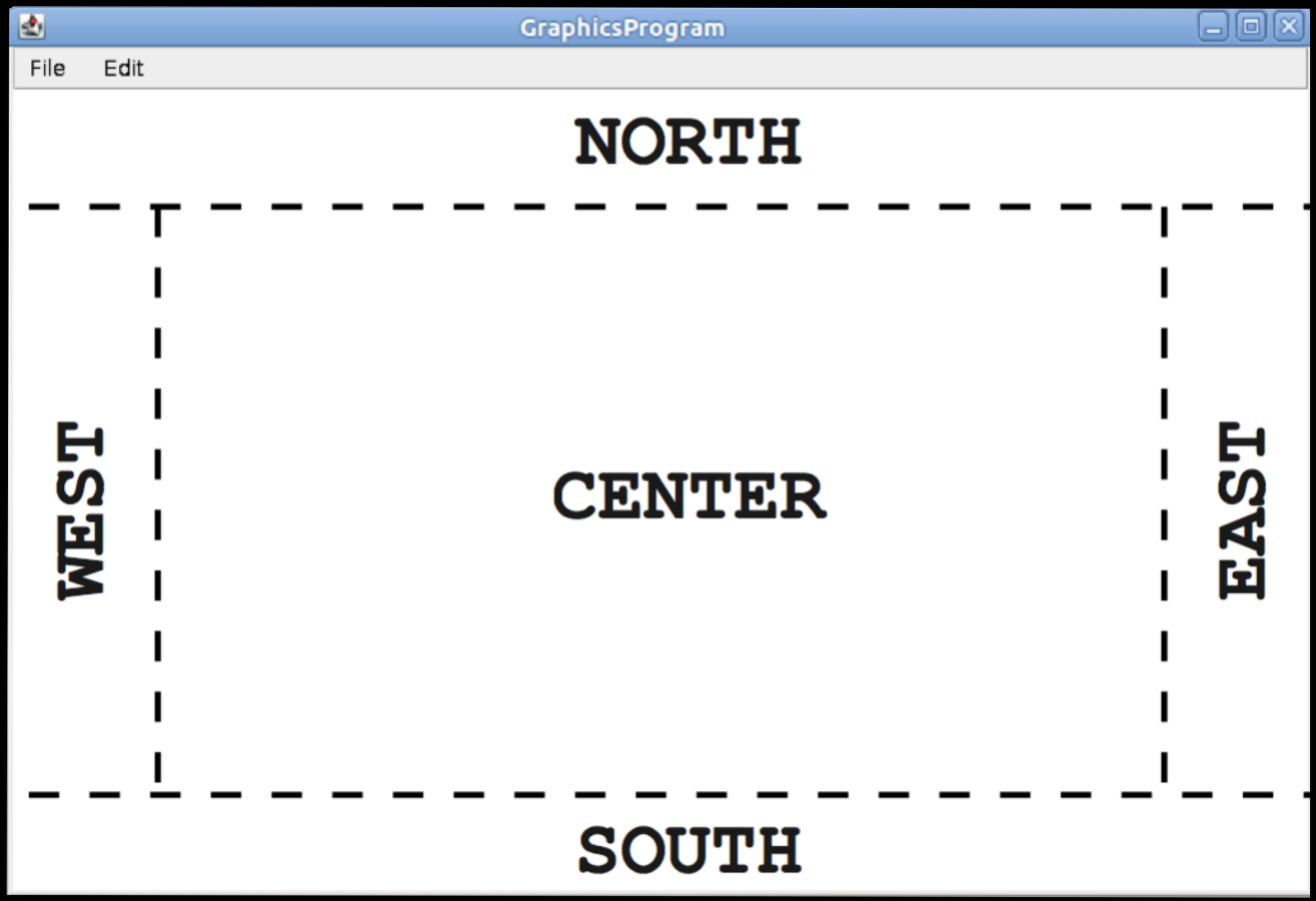
- Style doesn’t matter. You don’t need to comment.
  - (but…)
- Don’t worry about imports.
- Pseudocode credit is capped at 50%.

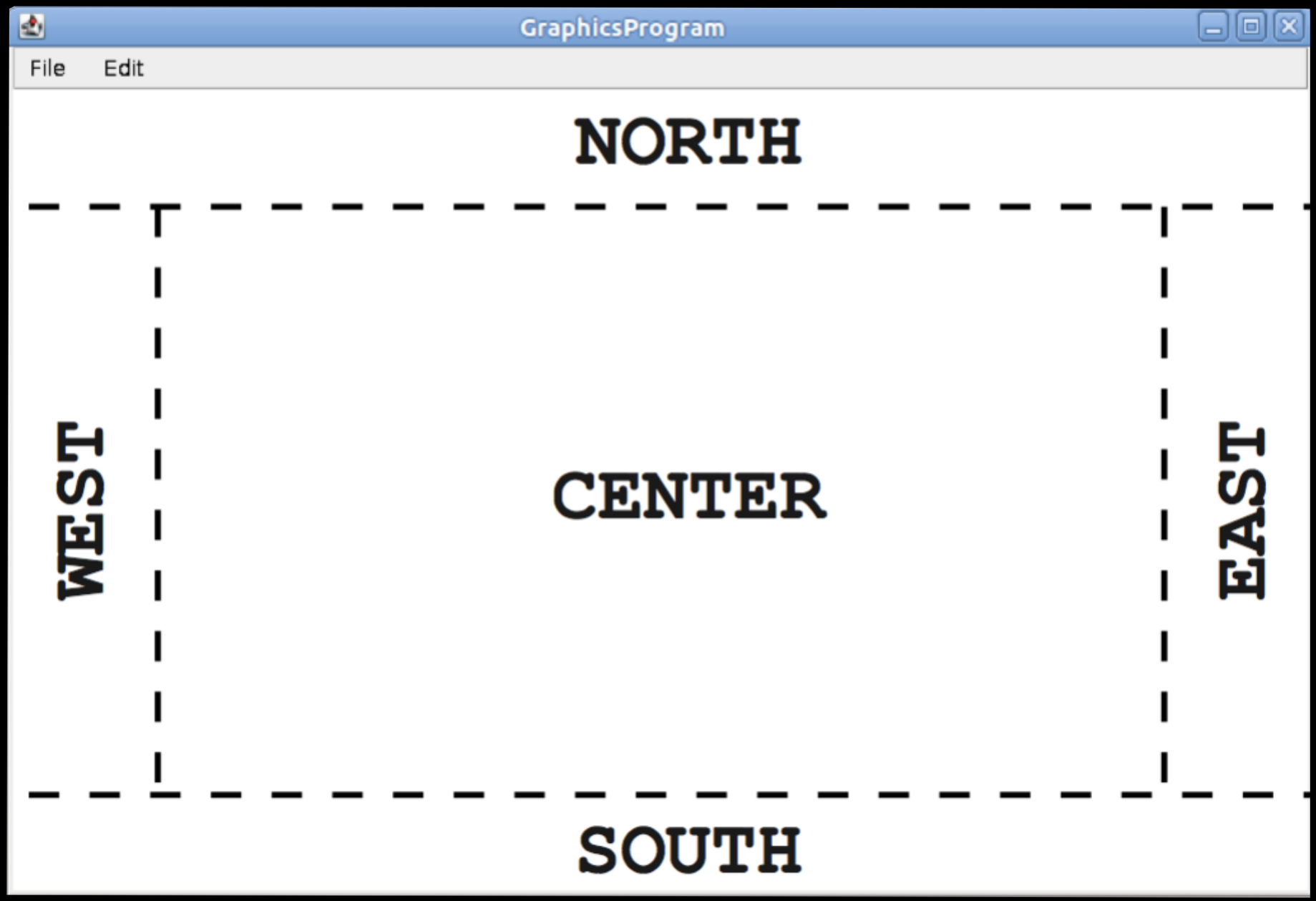
Other tips

- Style doesn’t matter. You don’t need to comment.
  - (but…)
- Don’t worry about imports.
- Pseudocode credit is capped at 50%.
- Edge-cases are really important.

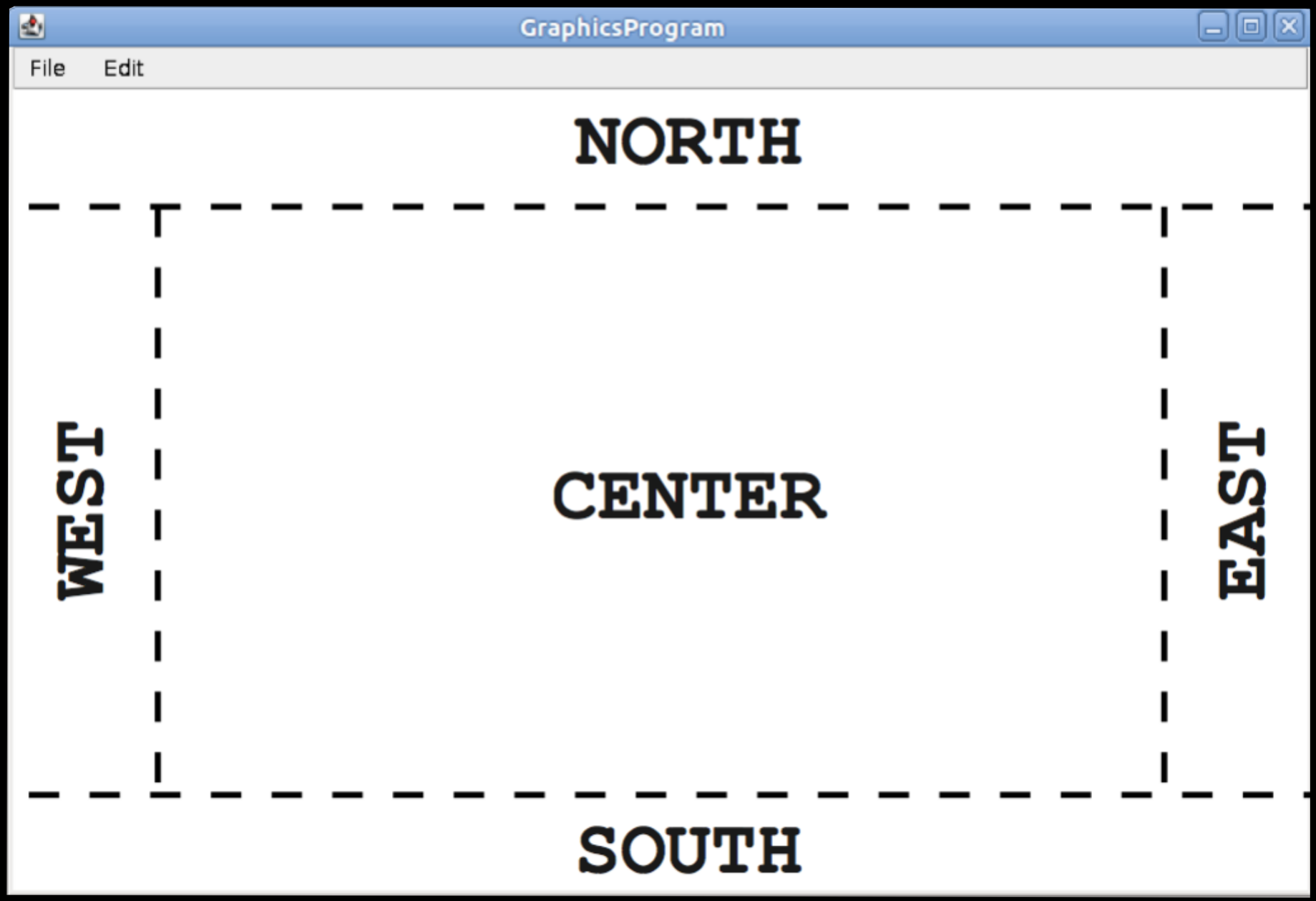
The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings


GraphicsProgram
File
Edit
NORTH
WEST
CENTER
EAST
SOUTH




```
JSlider slide = new JSlider(min, max, initial);
JButton button = new JButton("[button text]");
JTextField field = new JTextField("[field text]");
```




```
JSlider slide = new JSlider(min, max, initial);
JButton button = new JButton("[button text]");
JTextField field = new JTextField("[field text]");
add(interactor, location);
```

```
addActionListeners();
```

```
addActionListeners();
```

```
public void actionPerformed(ActionEvent e) {

}
```

```
addActionListeners();
```

```
public void actionPerformed(ActionEvent e) {

    if (e.getSource() == someInteractorIvar) {
        . . .
    }

}
```

```
addActionListeners();
```

```
public void actionPerformed(ActionEvent e) {

    if (e.getSource() == someInteractorIvar) {
        . . .
    }

}
```

requires instance variables

```
addActionListeners();
```

```
public void actionPerformed(ActionEvent e) {

    if (e.getSource() == someInteractorIvar) {
        . . .
    }

    if (e.getActionCommand().equals("[text in button]")) {
        . . .
    }

}
```

requires instance variables

```
addActionListeners();
```

```
public void actionPerformed(ActionEvent e) {

    if (e.getSource() == someInteractorIvar) {
        . . .
    }

    if (e.getActionCommand().equals("[text in button]")) {
        . . .
    }

}
```

requires instance variables

only works for buttons unless you use `someInteractor.setActionCommand("[action command]")`

How do I make a `JTextField` respond to the return key?

How do I make a `JTextField` respond to the return key?

```
private JTextField field;
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
    field = new JTextField(FIELD_WIDTH);
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
    field = new JTextField(FIELD_WIDTH);
    field.addActionListener(this);
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
    field = new JTextField(FIELD_WIDTH);
    field.addActionListener(this);
}
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
}

public actionPerformed(ActionEvent e) {
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
}

public actionPerformed(ActionEvent e) {
   if (e.getSource() == field)
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
}

public actionPerformed(ActionEvent e) {
   if (e.getSource() == field)
     doSomethingInResponseToEnter();
```

How do I make a `JTextField` respond to the return key?

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
}

public actionPerformed(ActionEvent e) {
   if (e.getSource() == field)
      doSomethingInResponseToEnter();
}
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
   field.setActionCommand(“field”);
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
   field.setActionCommand(“field”);
}
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
   field.setActionCommand(“field”);
}

public actionPerformed(ActionEvent e) {
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
   field.setActionCommand(“field”);
}

public actionPerformed(ActionEvent e) {
   if (e.getActionCommand().equals(“field”))
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
   field.setActionCommand(“field”);
}

public actionPerformed(ActionEvent e) {
   if (e.getActionCommand().equals(“field”))
     doSomethingInResponseToReturnKey();
```

How do I make a `JTextField` respond to the return key?

(version 2)

```
private JTextField field;

public init() {
   field = new JTextField(FIELD_WIDTH);
   field.addActionListener(this);
   field.setActionCommand(“field”);
}

public actionPerformed(ActionEvent e) {
   if (e.getActionCommand().equals(“field”))
     doSomethingInResponseToReturnKey();
}
```

In this problem, your task is to implement a simple graphical user interface for creating simple signs, each of which consists of lines of centered text displayed in different fonts. When you start the **`SignMaker`** program, the user interface looks like this, where the widths of the **`JTextField`** interactors are given by the constants **`CHARS_IN_LINE_FIELD`** and **`CHARS_IN_FONT_FIELD`**, respectively:

You can then add a line to the display by entering text in the field marked Line, as follows:

Hitting ENTER should add a new **`GLabel`** to the graphics display containing the text in the field. That **`GLabel`** should be centered in the window in both dimensions, adjusting the vertical position of the text by half the font ascent as you did for Assignment #2. The first line therefore is set in Times-Bold-36 as follows:

Note that hitting ENTER also clears the text field in the control strip making it easier for the user to enter the next line of the sign.

Once you have placed a label in the window, the user should be able to drag the label to another point on the screen. For example, the user might drag the label to the top, so the window looks like this:

The user can change the font of each line by typing a new font name in the Font field. For example, if the user wanted to add a second line to the message in a smaller, italic font, that user could do so by changing the contents of the Font field to Times-Italic-24 and then typing in the new message, like this:

Hitting ENTER at this point would add a new **`GLabel`** at the center of the window, like this:

You can then reposition either of these labels by clicking on them and dragging them to the desired location.

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {









   }


/* Instance variables */


/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {









   }
/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {


   }

/* Instance variables */


/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);









   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {


   }


/* Instance variables */
   private JTextField lineInputField;


/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);









   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {


   }


/* Instance variables */
   private JTextField lineInputField;


/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);









   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {

   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");









   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {

   }

/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);




   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {




   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);



   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {



   }

/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);


   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {


   }

/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);

   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {

   }

/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);
      addMouseListeners();
   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {

   }

/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);
      addMouseListeners();
   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == lineInputField) {



      }
   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);
      addMouseListeners();
   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == lineInputField) {
         GLabel label = new GLabel(lineInputField.getText());



      }
   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);
      addMouseListeners();
   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == lineInputField) {
         GLabel label = new GLabel(lineInputField.getText());
         label.setFont(fontField.getText());



      }
   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);
      addMouseListeners();
   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == lineInputField) {
         GLabel label = new GLabel(lineInputField.getText());
         label.setFont(fontField.getText());
         double x = (getWidth() - label.getWidth()) / 2;
         double y = (getHeight() + label.getAscent()) / 2;


      }
   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);
      addMouseListeners();
   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == lineInputField) {
         GLabel label = new GLabel(lineInputField.getText());
         label.setFont(fontField.getText());
         double x = (getWidth() - label.getWidth()) / 2;
         double y = (getHeight() + label.getAscent()) / 2;
         add(label, x, y);

      }
   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* SignMaker.java: this program allows the user to create a sign. */
public class SignMaker extends GraphicsProgram {

   public void init() {
      lineInputField = new JTextField(CHARS_IN_LINE_FIELD);
      lineInputField.addActionListener(this);
      fontField = new JTextField(CHARS_IN_FONT_FIELD);
      fontField.setText("Times-Bold-24");
      add(new JLabel("Line: "), SOUTH);
      add(lineInputField, SOUTH);
      add(new JLabel("  Font: "), SOUTH);
      add(fontField, SOUTH);
      addMouseListeners();
   }

/* Called when an action event occurs */
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == lineInputField) {
         GLabel label = new GLabel(lineInputField.getText());
         label.setFont(fontField.getText());
         double x = (getWidth() - label.getWidth()) / 2;
         double y = (getHeight() + label.getAscent()) / 2;
         add(label, x, y);
         lineInputField.setText("");
      }
   }


/* Instance variables */
   private JTextField lineInputField;
   private JTextField fontField;

/* Constants */
   private static final int CHARS_IN_LINE_FIELD = 30;
   private static final int CHARS_IN_FONT_FIELD = 15;
```

```
/* Instance variables */
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {


   }
```

```
/* Instance variables */
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {


   }
```

```
/* Instance variables */

   private GPoint last;
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());

   }
```

```
/* Instance variables */

   private GPoint last;
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());

   }
```

```
/* Instance variables */
   private GLabel dragLabel;
   private GPoint last;
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());
      dragLabel = (GLabel) getElementAt(last);
   }
```

```
/* Instance variables */
   private GLabel dragLabel;
   private GPoint last;
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());
      dragLabel = (GLabel) getElementAt(last);
   }

/* Called on mouse drag to reposition the object */
   public void mouseDragged(MouseEvent e) {



   }

/* Instance variables */
   private GLabel dragLabel;
   private GPoint last;
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());
      dragLabel = (GLabel) getElementAt(last);
   }

/* Called on mouse drag to reposition the object */
   public void mouseDragged(MouseEvent e) {
      if (dragLabel != null) {


      }
   }

/* Instance variables */
   private GLabel dragLabel;
   private GPoint last;
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());
      dragLabel = (GLabel) getElementAt(last);
   }

/* Called on mouse drag to reposition the object */
   public void mouseDragged(MouseEvent e) {
      if (dragLabel != null) {
         dragLabel.move(e.getX() - last.getX(), e.getY() - last.getY());

      }
   }

/* Instance variables */
   private GLabel dragLabel;
   private GPoint last;
```

```
/* Called on mouse press to record the coordinates of the click */
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());
      dragLabel = (GLabel) getElementAt(last);
   }

/* Called on mouse drag to reposition the object */
   public void mouseDragged(MouseEvent e) {
      if (dragLabel != null) {
         dragLabel.move(e.getX() - last.getX(), e.getY() - last.getY());
         last = new GPoint(e.getPoint());
      }
   }

/* Instance variables */
   private GLabel dragLabel;
   private GPoint last;
```

The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings

Problem 4: Arrays (25 points)

A *magic square* is an $n \times n$ grid of numbers with the following properties:

1. Each of the numbers 1, 2, 3, …, n^2 appears exactly once, and
2. The sum of each row and column is the same.

For example, here is a 3×3 magic square, which uses the numbers between 1 and $3^2 = 9$:

4	9	2
3	5	7
8	1	6

and here is a 5×5 magic square, which uses the numbers between 1 and $5^2 = 25$:

11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Write a method

```
private boolean isMagicSquare(int[][] square, int n);
```

that accepts as input a two-dimensional array of integers (which you can assume is of size $n \times n$) and returns whether or not it is a magic square.

```
private boolean isMagicSquare(int[][] matrix, int n) {









}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */









}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;









}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */









}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);









}

private int rowSum(int[][] grid, int row, int n) {



}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */









}

private int rowSum(int[][] grid, int row, int n) {



}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {




    }

}

private int rowSum(int[][] grid, int row, int n) {



}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)

        }
    }

}

private int rowSum(int[][] grid, int row, int n) {



}

private int colSum(int[][] grid, int col, int n) {



}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)
            return false;
        }
    }

}

private int rowSum(int[][] grid, int row, int n) {



}

private int colSum(int[][] grid, int col, int n) {



}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)
            return false;
        }
    }
    return true;
}

private int rowSum(int[][] grid, int row, int n) {



}

private int colSum(int[][] grid, int col, int n) {



}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)
            return false;
        }
    }
    return true;
}

private int rowSum(int[][] grid, int row, int n) {
    int sum = 0;



}

private int colSum(int[][] grid, int col, int n) {




}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)
            return false;
        }
    }
    return true;
}

private int rowSum(int[][] grid, int row, int n) {
    int sum = 0;
    for (int col = 0; col < n; col++) {

    }

}

private int colSum(int[][] grid, int col, int n) {




}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)
            return false;
        }
    }
    return true;
}

private int rowSum(int[][] grid, int row, int n) {
    int sum = 0;
    for (int col = 0; col < n; col++) {
        sum += grid[row][col];
    }

}

private int colSum(int[][] grid, int col, int n) {




}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)
            return false;
        }
    }
    return true;
}

private int rowSum(int[][] grid, int row, int n) {
    int sum = 0;
    for (int col = 0; col < n; col++) {
        sum += grid[row][col];
    }
    return sum;
}

private int colSum(int[][] grid, int col, int n) {




}
```

```
private boolean isMagicSquare(int[][] matrix, int n) {
    /* Make sure we see all numbers 1 to n * n. */
    if (!allExpectedNumbersFound(matrix, n)) return false;

    /* Sum up the first row to get its value. */
    int expected = rowSum(matrix, 0, n);

    /* Check that all rows and columns have this value. */
    for (int i = 0; i < n; i++) {
        if (rowSum(matrix, i, n) != expected ||
            colSum(matrix, i, n) != expected)
            return false;
        }
    }
    return true;
}

private int rowSum(int[][] grid, int row, int n) {
    int sum = 0;
    for (int col = 0; col < n; col++) {
        sum += grid[row][col];
    }
    return sum;
}

private int colSum(int[][] grid, int col, int n) {
    int sum = 0;
    for (int row = 0; row < n; row++) {
        sum += grid[row][col];
    }
    return sum;
}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {

}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];
```

```
}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {









            }
        }


}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */




            }
        }


}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
```

```
            }
        }
```

```
}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;




            }
        }
}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;

            /* Make sure it isn't used. */


        }
    }


}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;

            /* Make sure it isn't used. */
            if (used[square[row][col]])



        }
    }


}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;

            /* Make sure it isn't used. */
            if (used[square[row][col]])
                return false;



        }
    }


}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;

            /* Make sure it isn't used. */
            if (used[square[row][col]])
                return false;

            /* Mark the square used. */

        }
    }


}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;

            /* Make sure it isn't used. */
            if (used[square[row][col]])
                return false;

            /* Mark the square used. */
            used[square[row][col]] = true;
        }
    }


}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;

            /* Make sure it isn't used. */
            if (used[square[row][col]])
                return false;

            /* Mark the square used. */
            used[square[row][col]] = true;
        }
    }

    /* At this point, we know that all numbers are in range and there are
    * no duplicates, so everything is valid. */

}
```

```
private boolean allExpectedNumbersFound(int[][] square, int n) {
    boolean[] used = new boolean[n * n + 1];

    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            /* Make sure it's in range */
            if (square[row][col] < 1 || square[row][col] > n * n)
                return false;

            /* Make sure it isn't used. */
            if (used[square[row][col]])
                return false;

            /* Mark the square used. */
            used[square[row][col]] = true;
        }
    }

    /* At this point, we know that all numbers are in range and there are
    * no duplicates, so everything is valid. */
    return true;
}
```

The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings

`ArrayList` notes

`ArrayList` notes

- `ArrayList` is useful when you don't know how many entries you will have

`ArrayList` notes

- `ArrayList` is useful when you don't know how many entries you will have
- An `ArrayList` can be searched with `.contains()` which returns a `boolean`

`ArrayList` notes

- `ArrayList` is useful when you don't know how many entries you will have
- An `ArrayList` can be searched with `.contains()` which returns a `boolean`
- Another useful method is `.add(int index, data)` which shifts everything after index over by one so `data` can take that spot

`ArrayList` notes

- `ArrayList` is useful when you don't know how many entries you will have
- An `ArrayList` can be searched with `.contains()` which returns a `boolean`
- Another useful method is `.add(int index, data)` which shifts everything after index over by one so `data` can take that spot
- `.set(int index, data)` is used to replace the data at a given index

`ArrayList` notes

- **`ArrayList`** is useful when you don't know how many entries you will have
- An **`ArrayList`** can be searched with **`.contains()`** which returns a **`boolean`**
- Another useful method is **`.add(int index, data)`** which shifts everything after index over by one so **`data`** can take that spot
- **`.set(int index, data)`** is used to replace the data at a given index

0	1	2	3
"Hello"	"Hi"	"Hey"	"Yo"

`ArrayList` notes

- `ArrayList` is useful when you don't know how many entries you will have
- An `ArrayList` can be searched with `.contains()` which returns a `boolean`
- Another useful method is `.add(int index, data)` which shifts everything after index over by one so `data` can take that spot
- `.set(int index, data)` is used to replace the data at a given index

0	1	2	3
"Hello"	"Hi"	"Hey"	"Yo"

```
al.add(2, "Sup");
```

`ArrayList` notes

- `ArrayList` is useful when you don't know how many entries you will have
- An `ArrayList` can be searched with `.contains()` which returns a `boolean`
- Another useful method is `.add(int index, data)` which shifts everything after index over by one so `data` can take that spot
- `.set(int index, data)` is used to replace the data at a given index

0	1	2	3
"Hello"	"Hi"	"Hey"	"Yo"

```
al.add(2, "Sup");
```

0	1	2	3	4
"Hello"	"Hi"	"Sup"	"Hey"	"Yo"

How can I iterate across an `ArrayList<String> al`?

Option 1:

"for loop"

Option 2:

"iterator"

Option 3:

"for each"

How can I iterate across an `ArrayList<String> al`?

Option 1:

"for loop"

```
for (int i = 0; i < al.size(); i++)
```

Option 2:

"iterator"

Option 3:

"for each"

How can I iterate across an `ArrayList<String> al`?

Option 1:

"for loop"

```
for (int i = 0; i < al.size(); i++)
  println(al.get(i));
```

Option 2:

"iterator"

Option 3:

"for each"

How can I iterate across an `ArrayList<String> al`?

Option 1:

"for loop"

```
for (int i = 0; i < al.size(); i++)
  println(al.get(i));
```

Option 2:

"iterator"

```
Iterator<String> iter = al.iterator();
```

Option 3:

"for each"

How can I iterate across an `ArrayList<String> al`?

Option 1:

"for loop"

```
for (int i = 0; i < al.size(); i++)
  println(al.get(i));
```

Option 2:

"iterator"

```
Iterator<String> iter = al.iterator();
while(iter.hasNext())
```

Option 3:

"for each"

How can I iterate across an `ArrayList<String> al`?

Option 1:

"for loop"

```
for (int i = 0; i < al.size(); i++)
  println(al.get(i));
```

Option 2:

"iterator"

```
Iterator<String> iter = al.iterator();
while(iter.hasNext())
  println(iter.next());
```

Option 3:

"for each"

How can I iterate across an `ArrayList<String> al`?

Option 1:
"for loop"

```
for (int i = 0; i < al.size(); i++)
  println(al.get(i));
```

Option 2:
"iterator"

```
Iterator<String> iter = al.iterator();
while(iter.hasNext())
  println(iter.next());
```

Option 3:
"for each"

```
for (String str : al)
```

How can I iterate across an `ArrayList<String> al`?

Option 1:
"for loop"

```
for (int i = 0; i < al.size(); i++)
  println(al.get(i));
```

Option 2:
"iterator"

```
Iterator<String> iter = al.iterator();
while(iter.hasNext())
  println(iter.next());
```

Option 3:
"for each"

```
for (String str : al)
  println(str);
```

The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings

`HashMap` usage

HashMap usage

```
HashMap hm = new HashMap<KeyType, ValueType>();
```

HashMap usage

```
HashMap hm = new HashMap<String, String>();
```

HashMap usage

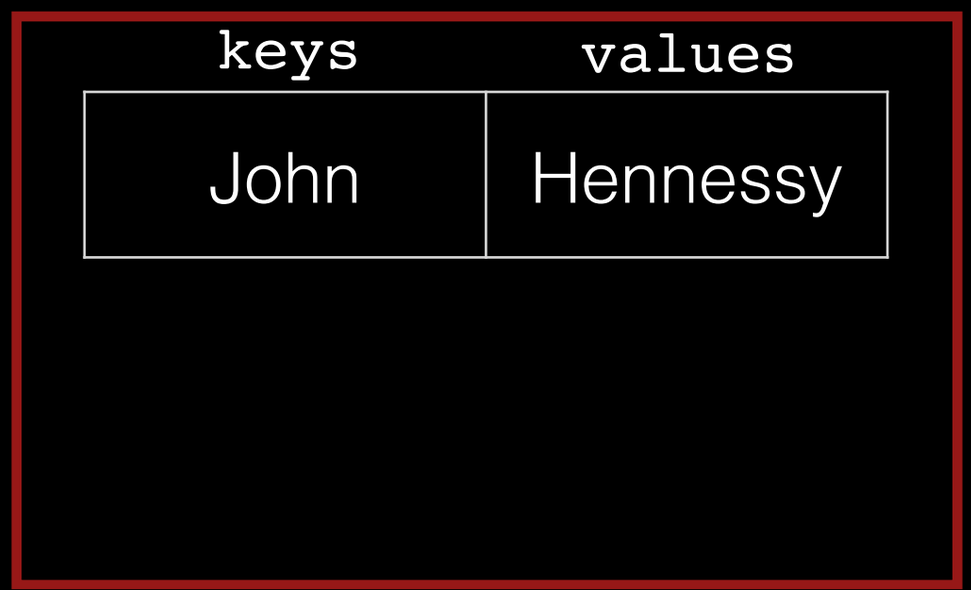
```
HashMap hm = new HashMap<String, String>();
```

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
```

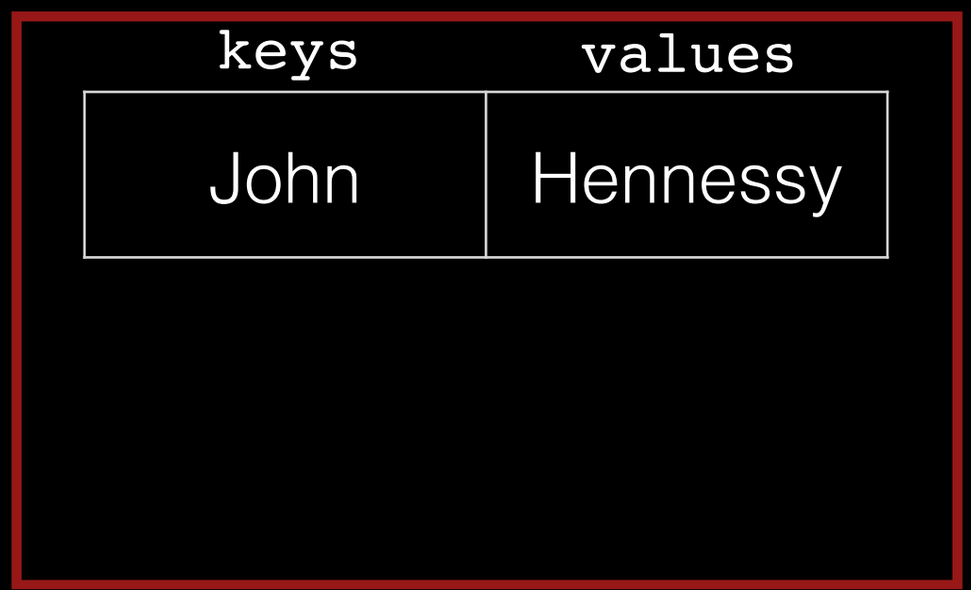
HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
```



HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);
```



HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);
```

keys	values
John	Hennessy
miles	seiver

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);
```

keys	values
John	Hennessy
miles	seiver

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);    //lastName is “seiver”
```

keys	values
John	Hennessy
miles	seiver

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);    //lastName is “seiver”

hm.put(“miles”, “miles”);
```

keys	values
John	Hennessy
miles	seiver

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);    //lastName is “seiver”

hm.put(“miles”, “miles”);
```

keys	values
John	Hennessy
miles	miles

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);    //lastName is “seiver”

hm.put(“miles”, “miles”);
```

```
lastName = hm.get(“miles”);
```

keys	values
John	Hennessy
miles	miles

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);    //lastName is “seiver”

hm.put(“miles”, “miles”);

lastName = hm.get(“miles”);           //lastName is “miles”
```

keys	values
John	Hennessy
miles	miles

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);    //lastName is “seiver”

hm.put(“miles”, “miles”);

lastName = hm.get(“miles”);           //lastName is “miles”

boolean in = hm.containsKey(“john”);
```

keys	values
John	Hennessy
miles	miles

HashMap usage

```
HashMap hm = new HashMap<String, String>();

hm.put(“John”, “Hennessy”);
hm.put(“miles”, “seiver”);

String lastName = hm.get(“miles”);    //lastName is “seiver”

hm.put(“miles”, “miles”);

lastName = hm.get(“miles”);           //lastName is “miles”

boolean in = hm.containsKey(“john”); //in is FALSE
```

keys	values
John	Hennessy
miles	miles

`HashMap` notes

`HashMap` notes

- If you call `get` on a key that doesn't exist, you will get `null`.

`HashMap` notes

- If you call `get` on a key that doesn't exist, you will get `null`.
- If you `put` a value for a key that already exists, the previous value is overwritten.

`HashMap` notes

- If you call `get` on a key that doesn't exist, you will get `null`.
- If you `put` a value for a key that already exists, the previous value is overwritten.
- A `HashMap` is unordered. The keys won't be iterated over in the same order you put them in.

How can I iterate across a `HashMap<String, Integer> hm`?

All the keys:

All the values:

How can I iterate across a `HashMap<String, Integer> hm`?

All the keys:

```
for (String key : hm.keySet())
```

All the values:

How can I iterate across a `HashMap<String, Integer> hm`?

All the keys:

```
for (String key : hm.keySet())
  println(key);
```

All the values:

How can I iterate across a `HashMap<String, Integer> hm`?

All the keys:

```
for (String key : hm.keySet())
  println(key);
```

All the values:

```
for (String key : hm.keySet())
```

How can I iterate across a `HashMap<String, Integer> hm`?

All the keys:

```
for (String key : hm.keySet())
  println(key);
```

All the values:

```
for (String key : hm.keySet())
  println(hm.get(key));
```

The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings

How do I find the size of each of these?

How do I find the size of each of these?

```
ArrayList<Integer> al?
```

How do I find the size of each of these?

```
ArrayList<Integer> al?   al.size()
```

How do I find the size of each of these?

```
ArrayList<Integer> al?   al.size()

char[] charArr?
```

How do I find the size of each of these?

```
ArrayList<Integer> al?   al.size()

char[] charArr?          charArr.length
```

How do I find the size of each of these?

`ArrayList<Integer> al?` `al.size()`

`char[] charArr?` `charArr.length`

`HashMap<String, Integer> hm?`

keys:

values:

How do I find the size of each of these?

`ArrayList<Integer> al?` `al.size()`

`char[] charArr?` `charArr.length`

`HashMap<String, Integer> hm?`

`hm.size()`

keys:

values:

How do I find the size of each of these?

`ArrayList<Integer> al?` `al.size()`

`char[] charArr?` `charArr.length`

`HashMap<String, Integer> hm?`

- # keys: `hm.size()`
- # values: `hm.size()`

How do I find the size of each of these?

`ArrayList<Integer> al?` `al.size()`

`char[] charArr?` `charArr.length`

`HashMap<String, Integer> hm?`

\# keys: `hm.size()`

\# values: `hm.size()`

`int[][] bigArr?`

\# rows:

\# columns:

How do I find the size of each of these?

```
ArrayList<Integer> al?          al.size()

char[] charArr?                 charArr.length

HashMap<String, Integer> hm?
    # keys:                     hm.size()
    # values:                   hm.size()

int[][] bigArr?
    # rows:                     bigArr.length
    # columns:
```

How do I find the size of each of these?

```
ArrayList<Integer> al?          al.size()

char[] charArr?                 charArr.length

HashMap<String, Integer> hm?
    # keys:                     hm.size()
    # values:                   hm.size()

int[][] bigArr?
    # rows:                     bigArr.length
    # columns:                  bigArr[0].length
```

Your task in this problem is to write a definition for a class called **`Localizer`** designed to help with the localization process. The constructor for the class has the form

```
public Localizer(String filename)
```

The constructor creates a new **`Localizer`** object and initializes it by reading the contents of the data file. The data file consists of an English word, followed by any number of lines of the form

xx=*translation*

where *xx* is a standardized two-letter language code, such as **`de`** for German, **`es`** for Spanish, and **`fr`** for French. Part of such a data file, therefore, might look like this:

`Localizations.txt`

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

Beyond the constructor, the only public method you need to define for **`Localizer`** is

```
public String localize(String word, String language)
```

```
public class Localizer {

    public Localizer(String filename) {

    }

    public String localize(String word, String lang) {

    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();









    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {

        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
```

```
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));

            while (true) {

            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));

            while (true) {
                String line = rd.readLine();
                if (line == null) break;
```

```
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));

            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
```

```
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));

            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
```

```
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
                else {

                    }
                }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
                else {
                    String lang = line.substring(0, equalSign);

                }
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
                else {
                    String lang = line.substring(0, equalSign);
                    String translation = line.substring(equalSign + 1);



                }
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
                else {
                    String lang = line.substring(0, equalSign);
                    String translation = line.substring(equalSign + 1);
                    if (map.get(word) == null)
                        map.put(word, new HashMap<String, String>());

                }
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
                else {
                    String lang = line.substring(0, equalSign);
                    String translation = line.substring(equalSign + 1);
                    if (map.get(word) == null)
                        map.put(word, new HashMap<String, String>());
                    map.get(word).put(lang, translation);
                }
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {


    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
                else {
                    String lang = line.substring(0, equalSign);
                    String translation = line.substring(equalSign + 1);
                    if (map.get(word) == null)
                        map.put(word, new HashMap<String, String>());
                    map.get(word).put(lang, translation);
                }
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {
        if (map.get(word) == null) return null;

    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

```
public class Localizer {

    private HashMap<String, HashMap<String, String>> map;

    public Localizer(String filename) {
        map = new HashMap<String, HashMap<String, String>>();
        try {
            BufferedReader rd = new BufferedReader(new FileReader(filename));
            String word = null;
            while (true) {
                String line = rd.readLine();
                if (line == null) break;
                int equalSign = line.indexOf('=');
                if (equalSign == -1)
                    word = line;
                else {
                    String lang = line.substring(0, equalSign);
                    String translation = line.substring(equalSign + 1);
                    if (map.get(word) == null)
                        map.put(word, new HashMap<String, String>());
                    map.get(word).put(lang, translation);
                }
            }
            rd.close();
        }
        catch (Exception ex) {
            throw new ErrorException(ex);
        }
    }

    public String localize(String word, String lang) {
        if (map.get(word) == null) return null;
        return map.get(word).get(lang);
    }
}
```

Localizations.txt

```
Cancel
de=Abbrechen
es=Cancelar
fr=Annuler
Close
de=Schließen
es=Cerrar
fr=Fermer
OK
fr=Approuver
Open
de=Öffnen
es=Abrir
fr=Ouvrir
```

The plan

- General info
- Graphics and interactivity
- 2D arrays
- ArrayLists
- HashMaps
- Data structure design
- Strings

How do I insert a letter in the middle of a `String`?

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;
```

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;

char toInsert = ‘C’;
```

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;
char toInsert = ‘C’;
int insertPos = 2;
```

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;
char toInsert = ‘C’;
int insertPos = 2;
String fixed =
```

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;
char toInsert = ‘C’;
int insertPos = 2;
String fixed =
   letters.substring(0, insertPos) +
```

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;
char toInsert = ‘C’;
int insertPos = 2;
String fixed =
   letters.substring(0, insertPos) +
   toInsert +
```

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;
char toInsert = ‘C’;
int insertPos = 2;
String fixed =
   letters.substring(0, insertPos) +
   toInsert +
   letters.substring(insertPos,
letters.length());
```

How do I insert a letter in the middle of a `String`?

```
String letters = “ABDE”;
char toInsert = ‘C’;
int insertPos = 2;
String fixed =
   letters.substring(0, insertPos) +
   toInsert +
   letters.substring(insertPos,
letters.length());
```

inclusive exclusive!

Your job in this problem is to write a predicate method

```
public boolean isAnagram(String s1, String s2)
```

that takes two strings and returns **`true`** if they contain exactly the same alphabetic characters, even though those characters may appear in any order. Thus, your method should return **`true`** for each of the following calls:

```
isAnagram("O, Draconian devil!", "Leonardo da Vinci")
isAnagram("Oh, lame saint!", "The Mona Lisa")
isAnagram("ALGORITHMICALLY", "logarithmically")
isAnagram("Doctor Who", "Torchwood")
```

These examples illustrate two important requirements for the **`isAnagram`** method:

- The implementation should look only at letters (i.e., characters for which the **`Character.isLetter`** method returns **`true`**), ignoring any extraneous spaces and punctuation marks thrown in along the way.
- The implementation should ignore the case of the letters in both strings.

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {




   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {

}
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];









      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }









      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {









      }
      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {
         char ch = str.charAt(i);




      }
      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {
         char ch = str.charAt(i);
         if (Character.isLetter(ch)) {

         }
      }
      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);









   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {
         char ch = str.charAt(i);
         if (Character.isLetter(ch)) {
            letterCounts[Character.toUpperCase(ch) - 'A']++;
         }
      }
      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);
      for (int i = 0; i < table1.length; i++) {

      }

   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {
         char ch = str.charAt(i);
         if (Character.isLetter(ch)) {
            letterCounts[Character.toUpperCase(ch) - 'A']++;
         }
      }
      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);
      for (int i = 0; i < table1.length; i++) {
         if (table1[i] != table2[i]) return false;
      }

   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {
         char ch = str.charAt(i);
         if (Character.isLetter(ch)) {
            letterCounts[Character.toUpperCase(ch) - 'A']++;
         }
      }
      return letterCounts;
   }
```

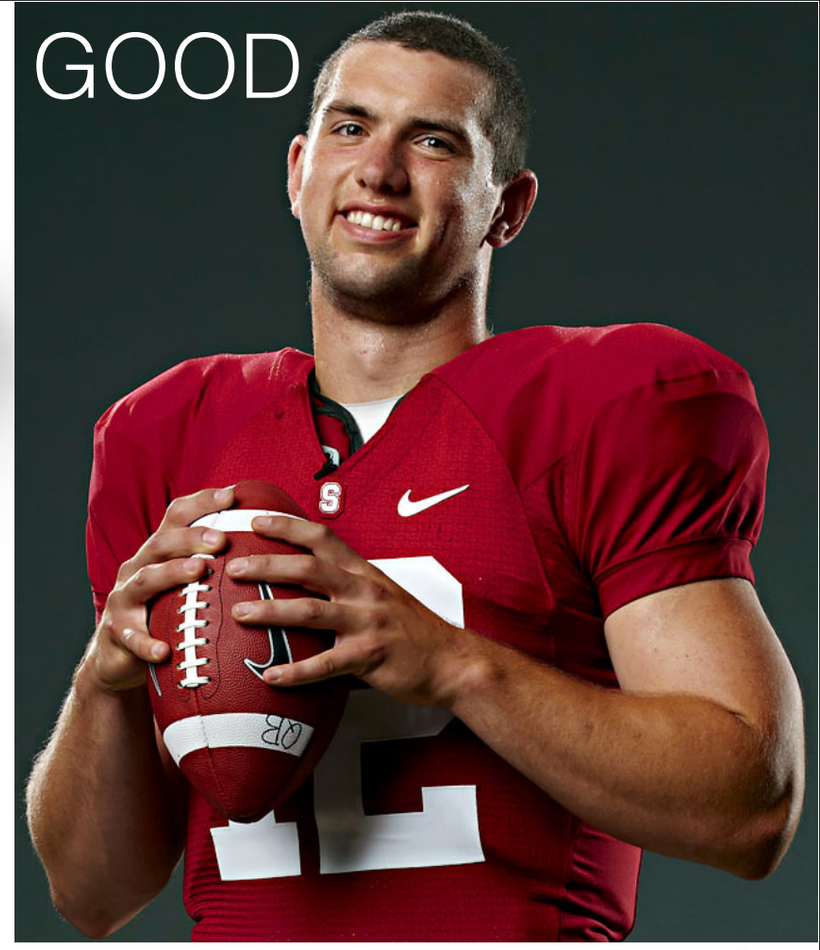
```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);
      for (int i = 0; i < table1.length; i++) {
         if (table1[i] != table2[i]) return false;
      }
      return true;
   }

/** Creates a letter-frequency table for the string */
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {
         char ch = str.charAt(i);
         if (Character.isLetter(ch)) {
            letterCounts[Character.toUpperCase(ch) - 'A']++;
         }
      }
      return letterCounts;
   }
```

```
/** Returns true if s1 and s2 are anagrams of each other */
   private boolean isAnagram(String s1, String s2) {
      int[] table1 = createFrequencyTable(s1);
      int[] table2 = createFrequencyTable(s2);
      for (int i = 0; i < table1.length; i++) {
         if (table1[i] != table2[i]) return false;
      }
      return true;
   }

/** Creates a letter-frequency table for the
   private int[] createFrequencyTable(String str) {
      int[] letterCounts = new int[26];
      for (char ch = 'A'; ch <= 'Z'; ch++) {
         letterCounts[ch - 'A'] = 0;
      }
      for (int i = 0; i < str.length(); i++) {
         char ch = str.charAt(i);
         if (Character.isLetter(ch)) {
            letterCounts[Character.toUpperCase(ch) - 'A']++;
         }
      }
      return letterCounts;
   }
```

Not strictly necessary since `int[]` is `0` by default


You know what...
yes you
CAN
!


SUCCESS


GOOD

Good Luck
4 ur
Exam

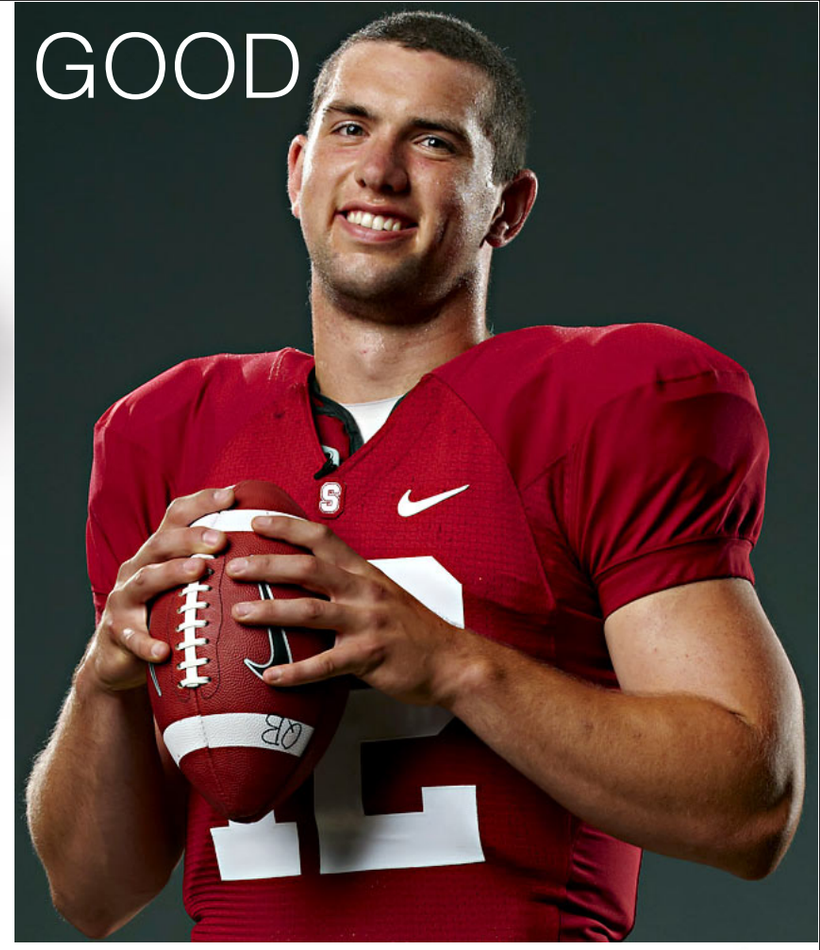

© GoodLightscraps.com
Best of Luck


Wishing
You
The
Best
of
Luck

Wish You a Good luck
SUCCESS


You know what...
yes you
CAN
!


SUCCESS


GOOD

Good Luck
4 ur
Exam


© GoodLightscraps.com
Best of Luck


Wishing
You
The
Best
of
Luck

Wish You a Good luck
SUCCESS