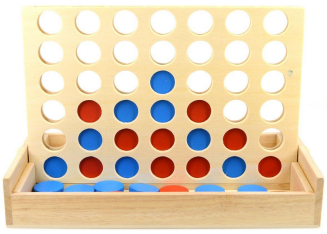
ChannelSurfer: Preparations (50 points)

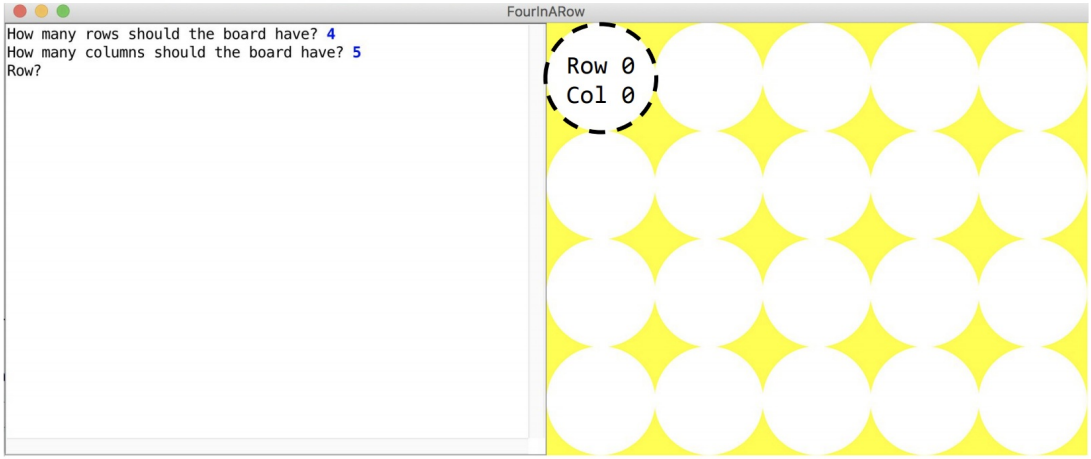


After a summer of rigorous studying, you return home ready to kick back and watch some TV with your best friend. Unfortunately, you've both already watched the new season of the show you had agreed to watch together! After a heated debate over what to watch instead, you decide to settle the matter with a game of wits.

Four-in-a-Row is a modified version of a classic (trademarked) game that you might have played before. The game proceeds with two players alternating turns, dropping one game piece into an upright board each turn. **In this version of the game**, the only way to win is by getting four pieces of your color connected **diagonally, with the diagonal going up and to the right**.

```
The first image below shows the initial game setup, with an empty
board represented by a yellow rectangle covered with white circles.
This board has 4 rows and 5 columns of spots, but the game can be
played with any board that has at least 4 rows and 4 columns.
The top left spot corresponds to row 0 and column 0.

The second image below shows the board state after one (valid) move,
with the spot at (row 3, column 1) filled in by a red piece.
```



Your friend, who also took a programming class this summer, already wrote some of the program. **You should read the starter code to understand what they have done so far.** Note that the program is decomposed into two classes: a `ConsoleProgram` for the game and

a `GCanvas` subclass for the board. They want you to implement a few of the more interesting methods for them:

Part A (10 points)

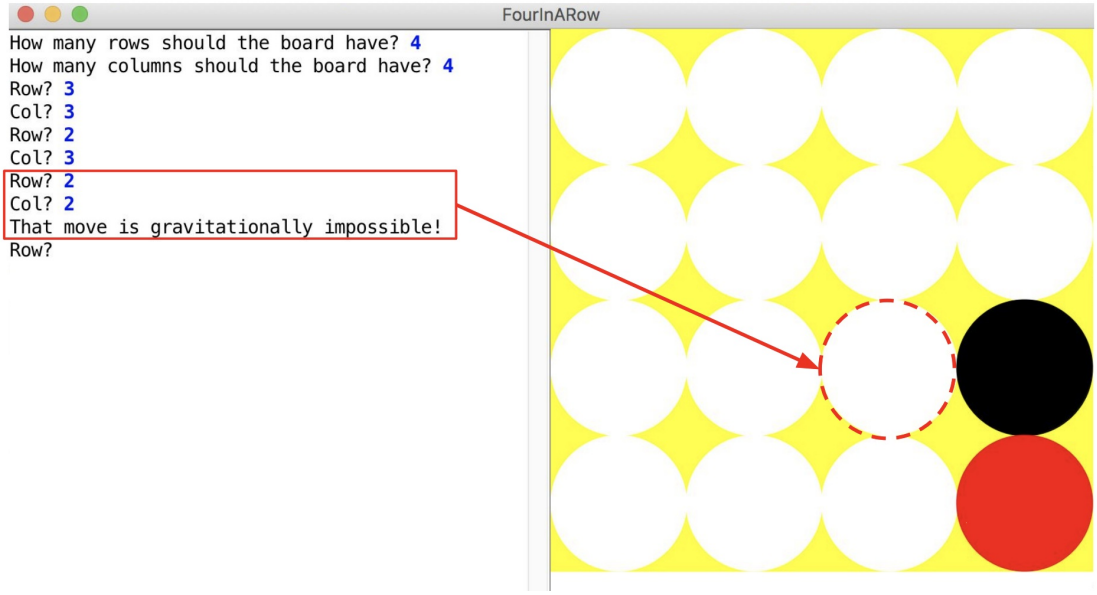
Write a `FourInARowBoard` method called `validateMove`, as follows:

```
public String validateMove(int row, int col)
```

In `validateMove`, you will check whether a player's move is allowed. You will return an error message if it is invalid and `null` if it is valid. You should make the following checks:

- Make sure the move is within the board's bounds. If not, the error message is `"That space is not on the board!"`
- Make sure the requested spot is empty (indicated by a `GOval` with the `EMPTY` color). If not, the error message is `"That space is already taken!"`
- Make sure the spot can actually support a piece; that is, it must be the lowest open spot in a column. If not, the error message is `"That move is gravitationally impossible!"` This point is illustrated in a diagram below.

```
For example, the space highlighted with a dashed circle is not a valid move,
because if you tried to put a piece there, it would fall through to a lower row.
```



Part B (5 points)

Write a `FourInARowBoard` method called `playPiece`, as follows:

```
public void playPiece(int row, int col, Color player)
```

In `playPiece`, you will update the board's internal representation to reflect a player having made a valid move. You should do so in such a way that the displayed board is updated in the process. Read your friend's code to make sure you understand how to make this happen.

Part C (10 points)

Write a `FourInARowGame` method called `takeTurn`, as follows:

```
private void takeTurn(Color player)
```

In `takeTurn`, you will first ask for the current player's next move, displaying the appropriate error message and reprompting on invalid moves until they enter a valid move. (There is an example of displaying an error message and reprompting in the diagram from Part A.) You will then make that move. *Hint*: You will find the methods from Parts A and B useful here.

Part D (25 points)

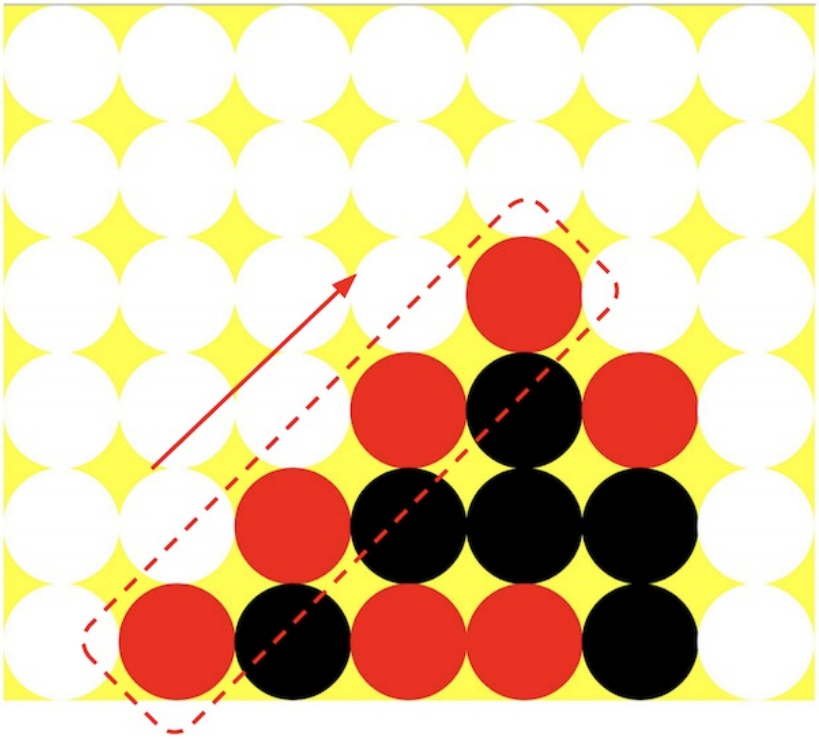
Write a `FourInARowBoard` method called `diagonalWin`, as follows:

```
private boolean diagonalWin()
```

In `diagonalWin`, you will check whether either player has won the game. You should return true if and only if there exists some uninterrupted streak of four pieces of the same color, all connected by a diagonal oriented as shown in the image below.

```
    The only way someone can win this game is by getting four connected pieces
of the same color in a diagonal oriented like the one circled with a dashed box.
    The arrow indicates the diagonal's direction (up and to the right).

  Getting four same-colored pieces in a row horizontally, vertically, or
    in a diagonal oriented up and to the left does not count as a win.
```



ChannelSurfer: HBO (25 points)

Having beaten your friend at Four-in-a-Row, you turn the TV on to HBO.

Issa works at We Got Y'all, an organization that provides educational after-school programs for youth in traditionally underserved communities. Eager to be recognized at work for her brilliance and unique skills, Issa volunteers to start tutoring the kids in computer science. Joanne loves the idea! The only problem is that Issa's computer science knowledge has grown a bit rusty since she took CS106A back at Stanford, and she is not about to ask Lawrence for help.

Issa is planning to take the kids on a trip to the zoo. Give Issa some talking points so that she can relate the zoo trip to programming.

Part A (5 points)

Explain how someone writing a zoo simulation might use **inheritance** to more easily model the behavior of the various zoo animals.

Assumptions and Specifications

- For the explanation above, you are **limited to 60 words at most**. Explanations that exceed 60 words will be penalized. **You do not have to write in complete sentences.**
- You may include other programming concepts in your answer, but you will not receive credit for a response that does not discuss inheritance.

Part B (5 points)

Animals at the zoo have different needs for what and when they eat. Explain how someone writing a program to efficiently schedule feedings might use a **`HashMap`** to help plan their route.

Assumptions and Specifications

- For the explanation above, you are **limited to 60 words at most**. Explanations that exceed 60 words will be penalized. **You do not have to write in complete sentences.**
- You do not have to describe how the route-planning program would work in detail. We are looking for:
  - What a key in this `HashMap` corresponds to
  - What a value in this `HashMap` corresponds to
  - How this `HashMap` would be used in the program (briefly)
- You may include other programming concepts in your answer, but you will not receive credit for a response that does not discuss a `HashMap`.
- Note that we are not looking for you to read our minds! There are lots of ways you could answer this question that would earn full credit. Just give one option that you think makes sense.

Part C (15 points)

Going over her old notes, Issa finds one algorithm labeled "IMPORTANT" but is having some trouble figuring out what it actually does. Write comments for Issa's program (on the right side of the screen, not directly in the code) to explain what's going on at three points in the code:

```
// 3. TODO explain what this method does
public void mystery(int[] arr) {
  int n = arr.length;

  for (int i = 0; i < n-1; i++) {
```

```
    int m = i;

    // 1. TODO explain what this for loop does
    for (int j = i+1; j < n; j++) {
      if (arr[j] < arr[m]) {
        m = j;
      }
    }

    // 2. TODO explain what this if statement does
    if (m != i) {
      int t = arr[i];
      arr[i] = arr[m];
      arr[m] = t;
    }
  }
}
```

Assumptions and Specifications

- For each of the explanations above, you are **limited to 30 words at most**. Explanations that exceed 30 words will be penalized. **You do not have to write in complete sentences.**
- Each explanation should address the high-level outcome of the entire specified code block, not just a single line. For example, the first explanation should describe what has happened after the for loop finishes, not just how many times the loop runs.
- *Hint*: If you are stuck, consider tracing through this program with a small sample `arr`. Tracing will take some time, but it should illuminate what is happening in the code.

ChannelSurfer: Bravo (40 points)

Your channel-surfing adventure continues with your friend's choice on Bravo.

It's makeover time on America's favorite TV show, but while Antoni is off slicing avocados, Bobby has been tasked with remodeling an entire house in just a few days! Give Bobby a hand by organizing his furniture options for quick lookup.

Write a class called `FurnitureCatalog`, as follows:

```
public class FurnitureCatalog
```

You will create a new `FurnitureCatalog` using a file of furniture items with their prices. Once created, a `FurnitureCatalog` should be able to tell Bobby two things: all of the price options for a specific item he's interested in, and a list of all of his "must-buys" (that is, items for which he has no choice because each has only one price option). `FurnitureCatalog` should also allow new items or new prices for existing items to be added on an individual basis, in case Bobby finds more options later.

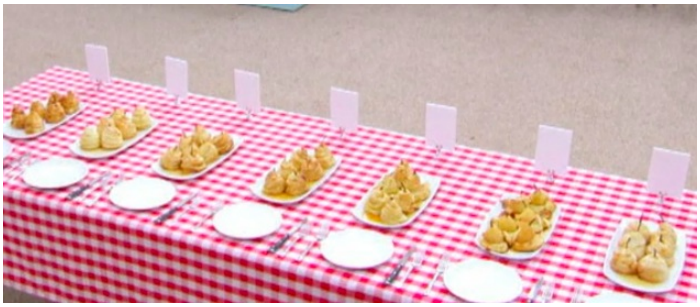
Assumptions and Specifications

- Part of what this problem assesses is your ability to choose the best data structure for a specific purpose. Make sure to think about what types of instance variables you could use in order to make the three public methods as simple as possible. You do not need the ideal solution for full credit, but strategies with significant redundancy or complexity will be penalized.
- You can assume that the provided file of furniture items is formatted so that each line is the name of an item (all lowercase), followed by a space, followed by its price. For example, a short file might look like this:

  ```
  mattress 499.99
  chair 49.99
  mattress 799.99
  ```

- You should catch file-reading exceptions where they might occur. It does not matter what you do upon catching an exception.
- The `getPriceOptions` method should be case-insensitive, i.e., Bobby should be able to look up "TaBLe" and get the price options for a table.
- If Bobby looks up a furniture item that is not in the catalog, `getPriceOptions` should return `null`.
- **`addItem` should silently do nothing if Bobby tries to add a price he has seen before for that item**, since the catalog already has that price as an option. If it is helpful, you can assume there were no duplicate prices in the original input file.

ChannelSurfer: PBS (65 points)



For the final show, you and your friend find a wholesome pick on PBS.

We're baking pies this week, and Paul doesn't want any soggy bottoms! For the technical challenge, Mary and Paul are judging the bakers on their ability to make the perfect poached pear puff pastry pies. Help the judges out by creating a GUI that they can use to rank the bakers on their pies.

Write an interactive program called `BakerRank`, as follows:

```
public class BakerRank extends GraphicsProgram
```

You will be provided an array of the bakers' names as a constant called `BAKERS`. Implementing the `BakerRank` program consists of three major tasks: adding buttons for ranking, creating a pie slideshow, and displaying the final results.

Buttons

You will create buttons that Mary and Paul will press to rank the pies. Specifically, you will create one button for each pie. Each button should display the word "Pie" followed by a space and then that pie's number in the lineup. The lineup's order is the same as the order of the names in the `BAKERS` array. For example, if the first baker in the array is Ruby, then the button for ranking Ruby's pie should display the text `"Pie 1"`. You should add the buttons to the `NORTH` region of your window. When reading the buttons from left to right, they should show up in order from 1 to the number of bakers. Each time Mary and Paul press a button corresponding to a pie, that pie's ranking will be set and pressing that button again should not have any effect. Mary and Paul will rank pies from **worst to best**. Once all pies have been ranked, you will display the bakers' rankings (see: Results).

Slideshow

You will display a slideshow that rotates, in order, through images of all the pies that have been baked. The slideshow will have an image for each baker's pie, where the name of each image is the name of the baker. For example, if there is a baker named Ali, then there will be an image named "Ali.jpg" for Ali's pie. You should resize your pie images so that they take up the entire screen (don't worry, the interactor regions will still show up). When cycling through the pies for the slideshow, each pie should be displayed for a whole second (you can use the `DELAY` constant of 1000 milliseconds for this). The slideshow can either keep going or stop when the ranking is finished; do whatever is easier. Note that the purpose of the slideshow is purely aesthetic; it does not have anything to do with the actual ranking process or interactors.

Results

After all buttons have been pressed, the rankings should be displayed in the `SOUTH` region of the window in order from best to worst. **Remember that Mary and Paul ranked the pies from worst to best, so the last button pressed is the top-ranked pie**. For example, if there are three bakers and Howard's pie's button is pressed first, Rob's pie's button is pressed second, and Kimberley's pie's button is pressed third, you should display "1: Kimberley 2: Rob 3: Howard".

```
The image on the left shows the program during the ranking phase.
   All buttons have been added, and the slideshow is playing.

The image on the right shows the program when rankings are done.
The bottom of the window shows the bakers ranked from best to worst,
which is the opposite of the order in which their buttons were pressed.
```

Assumptions and Specifications

- **For full credit, arrays should be the only type of data structure you use for this question. Solutions that use other types of data structures (including `ArrayList` and `HashMap`) will be eligible for at most 55 of the 65 possible points.**
- *As a reminder*, when you add multiple interactors to either the `NORTH` or the `SOUTH` region of the window, new interactors are added to the right of interactors that were already there.
- To reiterate, after a button is pressed once, pressing it again should do nothing. There are many ways to achieve the user-facing effect of "doing nothing". Figuring out any one of them might prove tricky, so it might help to ignore this requirement until the end.

*** CS 106A FINAL EXAM SYNTAX REFERENCE ***

This document lists some of the common methods and syntax that you will use on the exam.

Math (A&S 5.1)

```
double d = Math.pow(2, 5);  // 32.0
```

```
Math.abs(n),  Math.ceil(n),  Math.floor(n),  Math.log(n),  Math.log10(n),
Math.max(a, b), Math.min(a, b),  Math.pow(b, e),  Math.round(n),  Math.sqrt(n),
Math.sin(r), Math.cos(r), Math.tan(r),  Math.toDegrees(r),  Math.toRadians(d)
```

RandomGenerator (A&S 6.1)

```
RandomGenerator rg = RandomGenerator.getInstance();
```

`rg.nextBoolean()` `rg.nextBoolean(probability)`	returns a random `true`/`false` result; pass an optional probability from 0.0 - 1.0, or default to 0.5
`rg.nextColor()`	a randomly chosen `Color` object
`rg.nextDouble(min, max)`	returns a random real number between ***min*** and ***max***, inclusive
`rg.nextInt(min, max)`	returns a random integer between ***min*** and ***max***, inclusive

String (A&S Ch. 8)

```
String s = "hello";
```

`s.charAt(i)`	the character in this String at a given index
`s.contains(str)`	`true` if this String contains the other's characters inside it
`s.endsWith(str)`	`true` if this String ends with the other's characters
`s.equals(str)`	`true` if this String is the same as ***str***
`s.equalsIgnoreCase(str)`	`true` if this String is the same as ***str***, ignoring capitalization
`s.indexOf(str)`	first index in this String where given String begins (-1 if not found)
`s.lastIndexOf(str)`	last index in this String where given String begins (-1 if not found)
`s.length()`	number of characters in this String
`s.replace(s1, s2)`	a new string with all occurrences of *s1* changed to *s2*
`s.startsWith(str)`	`true` if this String begins with the other's characters
`s.substring(i, j)`	characters in this String from index ***i*** (inclusive) to ***j*** (exclusive)
`s.toLowerCase()` `s.toUpperCase()`	a new String with all lowercase or uppercase letters

Character/char (A&S Ch. 8)

```
char c = Character.toUpperCase(s.charAt(i));
```

`Character.isDigit(ch), .isLetter(ch),`
`.isLowerCase(ch), .isUpperCase(ch),`
`.isWhitespace(ch)`
	methods that accept a `char` and return `boolean` values of `true` or `false` to indicate whether the character is of the given type

`Character.toLowerCase(ch),` `.toUpperCase(ch)`	accepts a character and returns lower/uppercase version of it

Integer/int (A&S Ch. 8)

```
int num = Integer.parseInt("106");
```

`Integer.parseInt(String)`	accepts a numerical String and returns the value as an int

Scanner

```
Scanner input = new Scanner(new File("filename"));   // scan an input file
Scanner tokens = new Scanner(string);                // scan a string
```

`sc.next(),  sc.nextLine()`	read/return the next token (word) or entire line of input as a string
`sc.nextInt(),  sc.nextDouble()`	read/return the next token of input as an int or double
`sc.hasNext(),  sc.hasNextLine(),` `sc.hasNextInt(), sc.hasNextDouble()`	ask about whether a next token/line exists, or what type it is, without reading it
`sc.useDelimiter(String)`	set the character(s) on which the scanner breaks input into tokens
`sc.close()`	closes the scanner

Program, ConsoleProgram, GraphicsProgram

```
public class Name extends ProgramType { ... }
```

`init()`	executes before window appears; use to set up graphical components
`run()`	executes after window appears; use for animation loops, file loading, etc.

ConsoleProgram

```
public class Name extends ConsoleProgram { ... }
```

`readInt("prompt")`, `readDouble("prompt")`	Prompts/reprompts for a valid int or double, and returns it
`readLine("prompt");`	Prompts/reprompts for a valid String, and returns it
`readBoolean("prompt", "yesString", "noString");`	Prompts/reprompts for either ***yesString*** or ***noString*** (case-insensitive). Returns **true** if they enter ***yesString***, **false** if they enter ***noString***.
`promptUserForFile("prompt", "directory");`	Prompts for a filename, re-prompting until input is a file that exists in the given directory. Returns the full file path (***"directory/filename"***).
`println("text");`	Prints the given text to the console, followed by a newline ('\n').
`print("text");`	Prints the given text to the console.

GraphicsProgram

```
public class Name extends GraphicsProgram { ... }
```

`add(shape), add(shape, x, y);`	displays the given graphical shape/object in the window (at **x**, **y**)
`getElementAt(x, y)`	returns graphical object at the given x/y position, if any (else `null`)
`getHeight(), getWidth()`	the height and width of the graphical window, in pixels
`pause(ms);`	halts for the given # of milliseconds
`remove(shape);`	removes the graphical shape/object from window so it will not be seen
`setCanvasSize(w, h);`	sets canvas's onscreen size
`setBackground(color);`	sets canvas background color

Graphical Objects (A&S Ch. 9)

```
GRect rect = new GRect(10, 20, 50, 70);
```

`new GLabel("text", x, y)`	text with bottom-left at (x, y)
`new GLine(x1, y1, x2, y2)`	line between points (x1, y1), (x2, y2)
`new GOval(x, y, w, h)`	largest oval that fits in a box of size w * h with top-left at (x, y)
`new GRect(x, y, w, h)`	rectangle of size w * h with top-left at (x, y)
`obj.getColor(), obj.getFillColor()`	returns the color used to color the shape outline or interior
`obj.getX(), obj.getY(), obj.getWidth(), obj.getHeight()`	returns the left x, top y coordinates, width, and height of the shape
`obj.move(dx, dy);`	adjusts location by the given amount
`obj.setFilled(boolean);`	whether to fill the shape with color
`obj.setFillColor(Color);`	what color to fill the shape with
`obj.setColor(Color);`	what color to outline the shape with
`obj.setLocation(x, y);`	change the object's x/y position
`obj.setSize(w, h);`	change the object's width and height
`label.setLabel(String);`	changes the text that a GLabel displays
`label.getAscent(),label.getDescent()`	returns a GLabel's ascent or descent from the baseline
`new GImage("filename", x, y)`	image from the given file, drawn at (x, y)
`new GImage(pixelArray)`	image from the given 2D array of `int` pixels
`image.getPixelArray(), setPixelArray(a)`	return/set 2D array of `int`s representing pixels of the image
`GImage.getRed(px)`, `getGreen(px)`, `getBlue(px)`	returns the individual red/green/blue components of a given `int` pixel
`GImage.createRGBPixel(r, g, b)`	creates and returns an `int` pixel with the given r/g/b values
`GImage.createRGBPixel(r, g, b, a)`	creates and returns an `int` pixel with the given r/g/b/alpha values

Colors

```
rect.setColor(Color.BLUE);
```

```
Color.BLACK, BLUE, CYAN, GRAY, GREEN, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW
Color name = new Color(r, g, b);     // red, green, blue from 0-255
```

Mouse Events (A&S Ch. 10)

```
public void eventMethodName(MouseEvent event) { ...
```

events: `mouseMoved`, `mouseDragged`, `mousePressed`, `mouseReleased`, `mouseClicked`, `mouseEntered`, `mouseExited`

`e.getX(), e.getY()`	the x or y-coordinate of mouse cursor in the window

Array (A&S Ch. 11)

```
int[] arr = new int[5];          int[][] pixels = new int[5][2];
```

`new type[length]`	creates a new 1D array of the given type and length
`new type[rows][cols]`	creates a new 2D array of the given type and number of rows and cols
`arr[i], arr[i][j], ...`	returns the element at index i, index (i,j), etc.
`arr.length`	returns the length of the array
`Arrays.toString(arr)`	returns a string representing the array, such as ***"[10, 30, -25, 17]"***
`Arrays.sort(arr)`	sorts the elements in place (no return value)
`Arrays.equals(arr1, arr2)`	returns true if the arrays contain the same elements in the same order
`Arrays.fill(arr, value)`	sets every element to the given value
`Arrays.deepToString(arr)`	returns a string representing the multidimensional array, such as ***"[[0, 1, 2], [1, 2, 3], [2, 3, 4]]"***
`Arrays.deepEquals(arr1, arr2)`	returns true if the multidimensional arrays contain the same elements in the same order.

ArrayList (11.8)

```
ArrayList<Integer> list = new ArrayList<>();
```

`L.add(value);` `L.add(index, val);`	append to end of list; or insert at index, shifting right
`L.clear();`	removes all elements
`L.contains(value)`	`true` if value is in the list
`L.equals(L2)`	`true` if same elements
`L.get(index)`	returns value at given index
`L.indexOf(value)` `L.lastIndexOf(val)`	first/last index where given value is found (or -1 if not found)
`L.isEmpty()`	`true` if the list has no elements
`L.remove(index);`	removes value at given index, shifting subsequent values left
`L.remove(val);`	removes first occurrence of value
`L.set(index, val);`	replaces value at given index
`L.size()`	number of elements in the list
`L.toString()`	string representation of list such as `"[10, -2, 43]"`

HashMap (13.2)

```
HashMap<String, Double> map = new HashMap<>();
```

`M.put(key, value);`	adds a pair between the given key and value, replacing any old pair for that key
`M.clear();`	removes all elements
`M.containsKey(key)`	returns `true` if the given key is a key of a pair in this map
`M.equals(map2)`	`true` if same key/value pairs
`M.get(key)`	returns value paired with key, or `null`
`M.keySet()`	a collection of all keys in the map
`M.isEmpty()`	`true` if the map contains no pairs
`M.remove(key);`	removes pair for the given key, if there is one; does nothing if not
`M.values()`	collection of all values in map
`M.size()`	returns number of pairs in map
`M.toString()`	returns a string representation such as `"{a=b, c=d, e=f}"`

```
// collection is a HashMap key/value set, array, or ArrayList
for (type name : collection) { ...
```

Interactors (A&S 10.5-10.6)

```
JButton button = new JButton("Click me!");
```

`new JButton("text")`	button displaying the given text
`addActionListeners()`	sets up program to hear action events on all added buttons
`new JLabel("text")`	label displaying the given text
`new JTextField(width)`	text field with the given width (in characters)
`textField.addActionListener(this)`	sets up program to hear an action event when ENTER key typed
`.getText(), .setText(text)`	get/set the text being displayed in the button/label/text field
`add(component, region)`	adds the given interactor in the given window region (e.g. SOUTH)
`.setActionCommand("text"), .getActionCommand()`	gets/sets the action command associated with an interactor.

```
public void actionPerformed(ActionEvent event) { ...
```

`e.getActionCommand()`	action command of the triggered interactor (e.g. text of clicked button)
`e.getSource()`	the triggered component/interactor itself