Georgia Tech

Object-Oriented Programming

CS 1331

Recitation 12

Announcements

- Homework 15 is due Wednesday, November 20th
- Exam 3 is on Friday November 22!!
    - Let your professor know if you will need a make-ahead NOW if you haven’t already

Generics

- Allows for **type parameterization**
  - type parameters must be Objects, not primitive types
- You supply the type when you instantiate a generic class
  - Ex. `ArrayList<Integer> arr = new Arraylist<>();`
- Cannot instantiate a variable or array of a generic type
  - To make a generic array cast an Object array to type T[]
- Type Bounds:
  - Use `extends` for upper bounds:
    - Ex. `<T extends SuperClassName & InterfaceName>`
  - Use `super` for lower bounds:
    - Ex. `<? super T>`

Generic Classes

- Generic types can be used for return types and static types of variables within a generic class or interface
  - Ex.

```
class Tester<T> {
    T var;
    public Tester(T var) { this.var = var; }
    public T getVar() { return var; }
}
```

- Convention to use uppercase letters (E, K , V, N , T)
- **type erasure** - generic parameters are replaced with actual classes at compile time

Searching

- **Linear search** - check every element in order
  - Simple but O(n) time
- **Binary search**
  1. Find midpoint of array (`midpoint = (begin + end) / 2`)
  2. if `arr[midpoint] == key` -> return

     else if `arr[midpoint] < key` -> recurse on right half of array

     else if `arr[midpoint] > key` -> recurse on left half of array

  - Only works on sorted arrays
  - O(log n) time

Asymptotics/Big O Notation

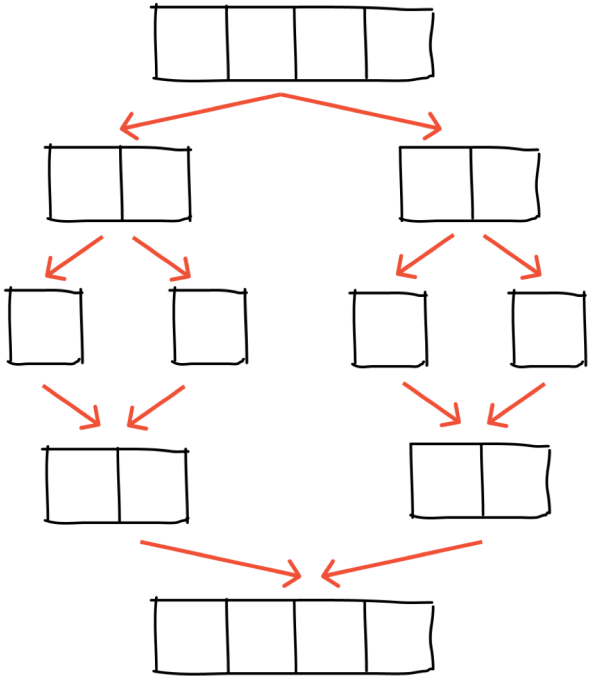
- Measure of efficiency of code; describes the rate at which the number of operations performed increases *relative to the size of the input*
- Notation for this rate is O(x) where x is some order of the input size *n*
  - $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)$
  - We can ignore constants when expressing Big O time complexity

Sorting

- **Bubble sort** - repeatedly compares adjacent elements and swaps if necessary
  - After n passes the last n elements are sorted and in their final position
  - Algorithm ends when an entire pass is made without any swaps
- **Selection sort** - selects min of unsorted elements and moves to front of unsorted subarray
  - After n passes the first n elements are sorted and in their final position
- **Insertion sort** - bubbles the $(n + 1)^{th}$ element backwards on the n^{th} pass
  - After n passes the first n + 1 elements are sorted, but not necessarily in their final position

Merge Sort

- Base case: array length of 1
- Recursive call: call `mergeSort()` on left and right halves of the array
- Call helper function `merge(int arr[], int l, int m, int r)` to merge the two halves of the array back together



Big O of Sorts

Selection Sort	$O(n^2)$
Bubble Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Merge Sort	$O(n \log n)$

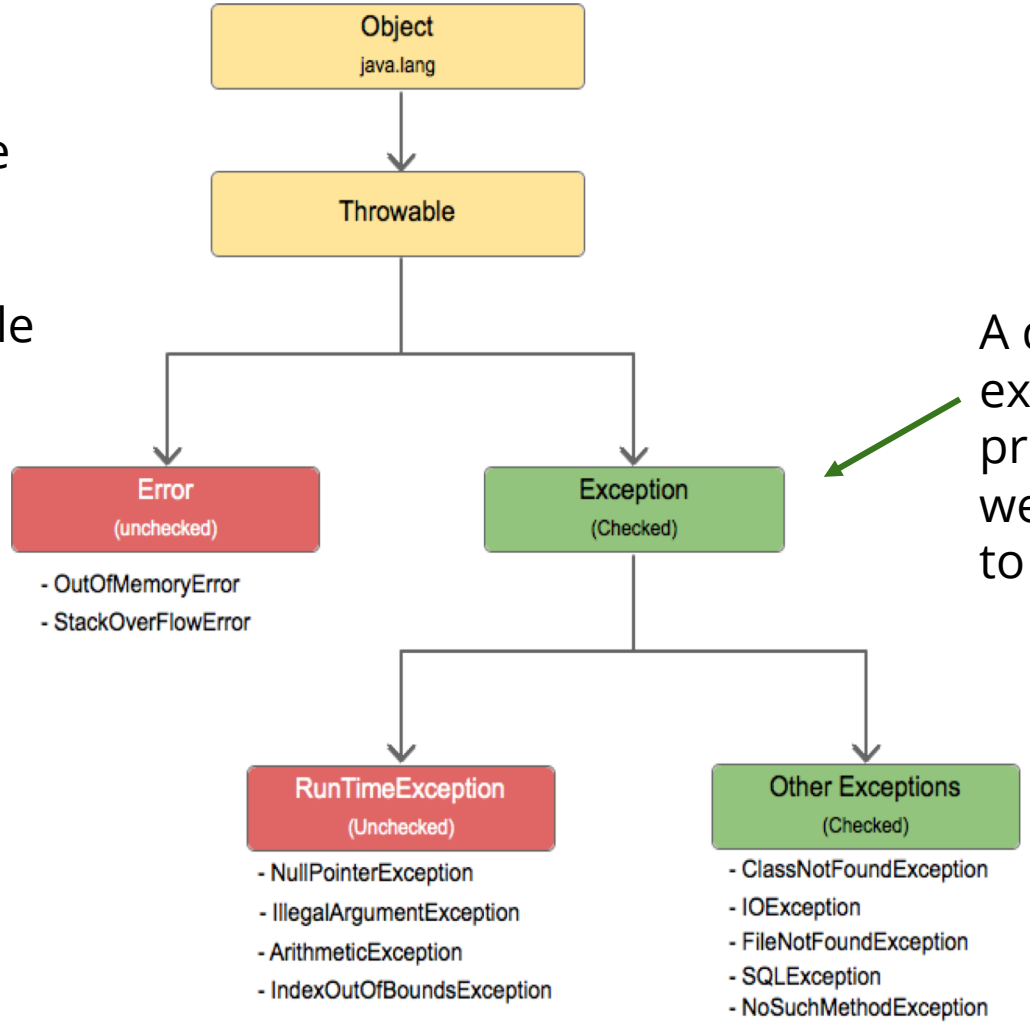
Recursion

- Approach to solving problems that breaks it up into smaller parts
- 3 important parts of a recursive method:
  - ***Base case*:** terminating condition
  - **Recursive call:** the function calls itself with new parameters
  - **Converges to base case:** every time the method is called, it should be solving a problem that's closer to the base case than the previous time
    - If the base case is never reached or defined **stack overflow** will occur

Exceptions

- Exception: something that went wrong during the execution of our program, and disrupts it from continuing. All exceptions extend from `Throwable`.
- There are two type of exceptions:
  - Checked: compiler **won't allow code to compile** if these are not handled, either by catching them or declaring them to be thrown.
    - Examples: `IOException`, `FileNotFoundException`
  - Unchecked: not checked by compiler, all extend `RuntimeException`
    - These are typically just a result of bugs in the code
    - Examples: `NullPointerException`, `ArithmeticException`

Something more serious that our program should NOT try to handle



A disruption in the execution of our program for which we can write code to handle

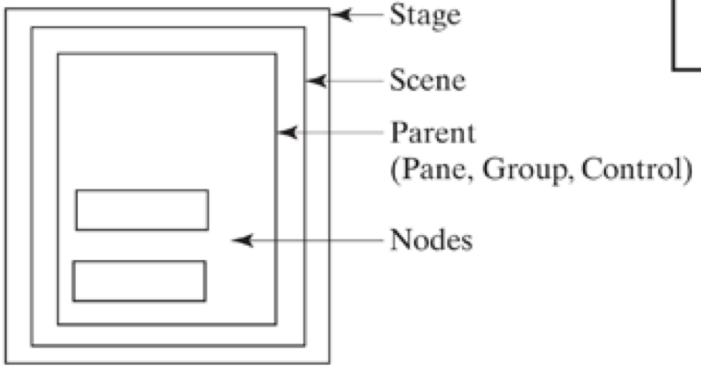
Handling Exceptions

- What to do?
  1. Nothing - not allowed for checked exceptions
  2. Try catch block - handle it right away
     - Chain catch blocks from most specific Exception to most general
  1. Throw to another method - delegate handling of Exception
     - `throws` is used in the method header to propagate a potential Exception
     - `throw` is used in the method body when an Exception is explicitly thrown
- If a method makes a call to another method that throws an Exception, the calling method must either handle the Exception with a try catch block or declare in its method header that it also throws the Exception

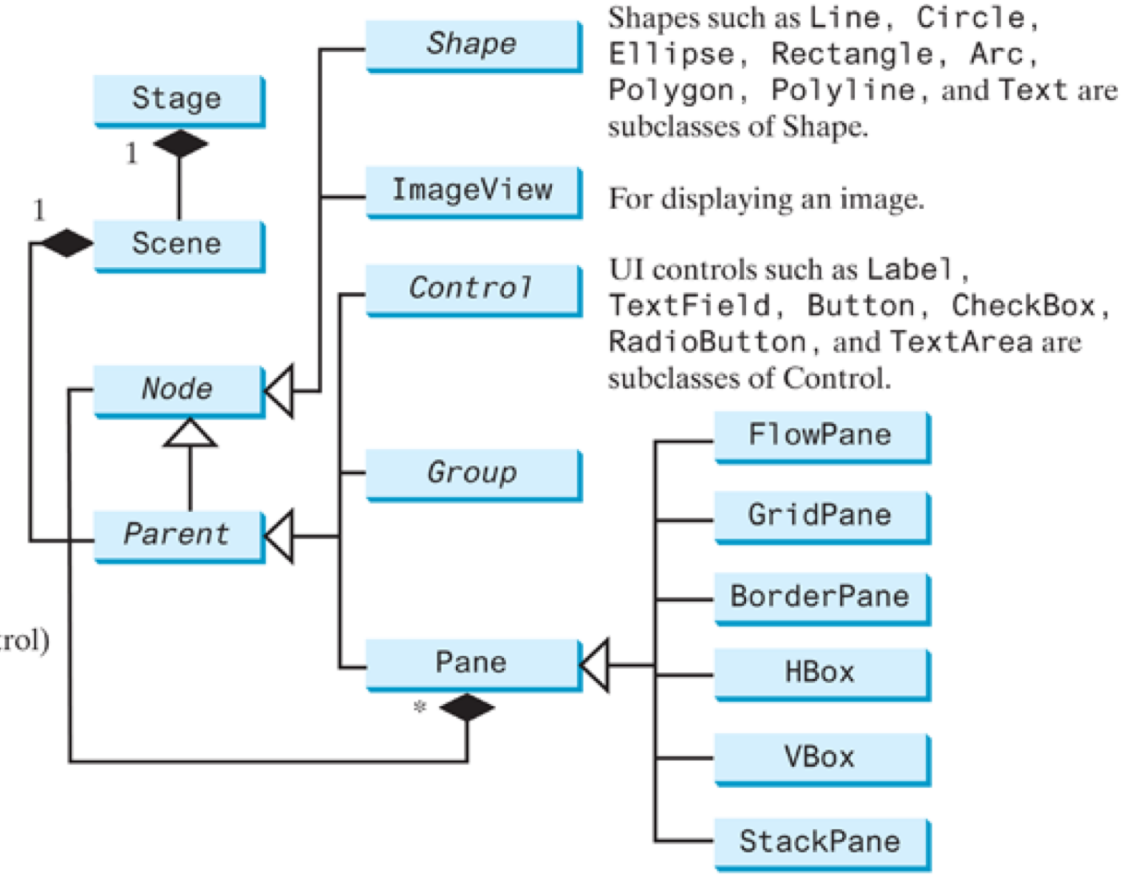
File I/O

- Writer
  - PrintWriter
    - Different constructors for taking in a String or File object as parameter
    - Uses familiar `print`, `println`, `printf` methods
  - FileWriter
    - Better for outputting raw data
- open(), flush() & close() file
  - ~~`open()`: Selects the file that you want to write to~~
  - `flush()`: Writes the file to disk from memory
  - `close()`: Finishes the edits and exits the File

JavaFX


Stage
Scene
Parent
(Pane, Group, Control)
Nodes

(a)


Shape
Shapes such as Line, Circle, Ellipse, Rectangle, Arc, Polygon, Polyline, and Text are subclasses of Shape.
Stage
1
1
Scene
ImageView
For displaying an image.
Control
UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.
Node
Group
Parent
FlowPane
GridPane
BorderPane
HBox
Pane
*
VBox
StackPane

(b)

JavaFX Common Layouts and Nodes

- HBox (Horizontal Box)
  - Formats all the children, or items, stored within the HBox and is stored horizontally from left to right
- VBox (Vertical Box)
  - Like HBox but rather than displaying elements left to right, it does it top down
- Label
  - Displays text
- Button
  - A clickable button
- RadioButton
  - A button that can be selected or not selected

Event Driven Programming

- “Events” is something that occurs in the programs that is intended to trigger another action to happen
- Event delegation: The program in charge of triggering the event is not in charge of the implementation for it.

```
class PressedHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Button Pressed from Inner Class!");
    }
}
```

Inner Classes

```
public void start (Stage primaryStage) {
    Button bt = new Button("Button 1");
    Scene sc = new Scene(bt);
    primaryStage.setScene(sc);
    primaryStage.show();

    bt.setOnAction(new PressedHandler());
}

class PressedHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Button Pressed from Inner Class!");
    }
}
```

Anonymous Inner Classes & Lambdas

```
bt.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Button pressed from Anonymous Class!");
    }
});
```

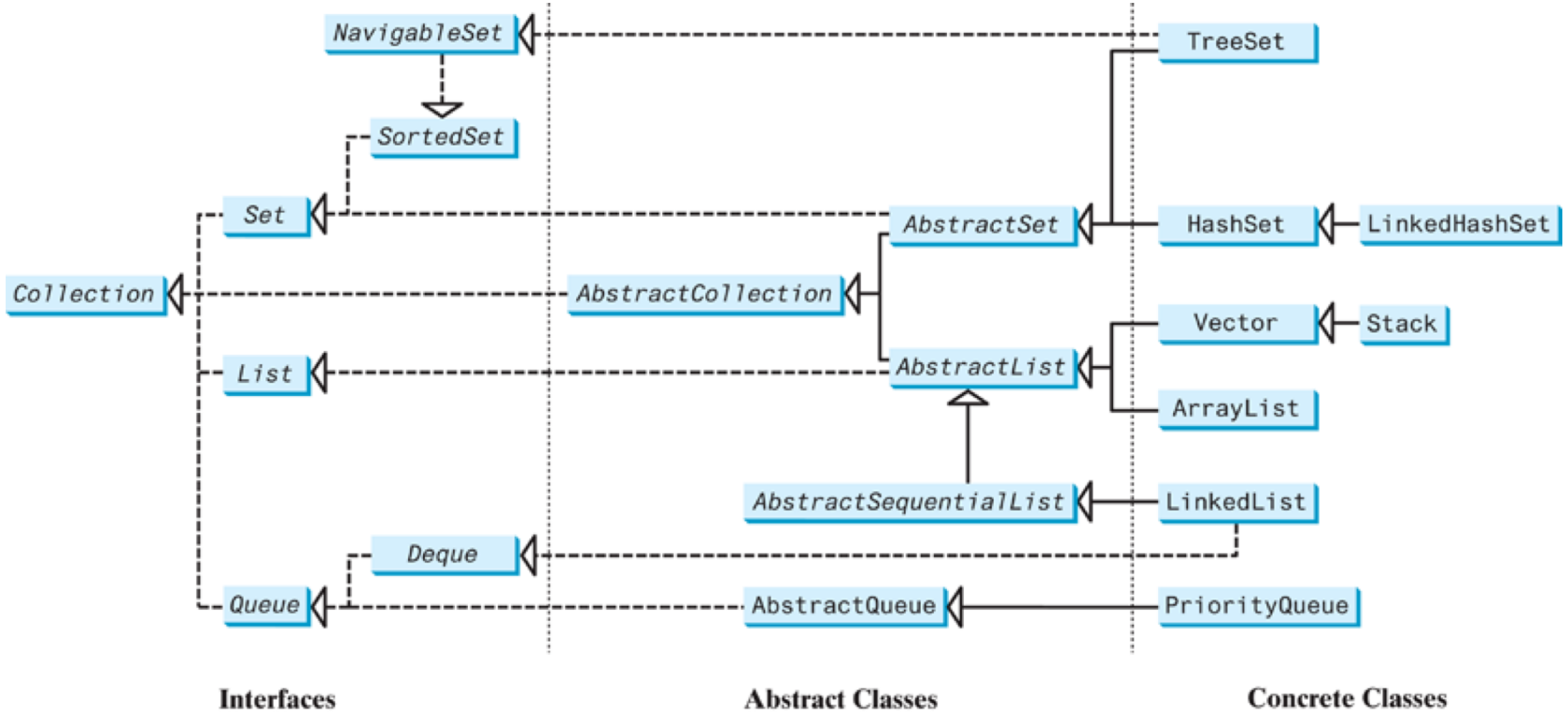
```
bt.setOnAction((ActionEvent e) -> {
    System.out.println("Button Pressed from Lambda!");
});

bt.setOnAction(e -> {
    System.out.println("Button Pressed from Lambda!");
});
```

Abstract Data Types

- An **abstract data type** (ADT) is a data type whose creation and update are constrained to specific well-defined operations. A class can be used to implement an ADT.
- Java has built in ADT and Data structures for you to use. The basic/base ADT is the Collection<E> interface.
- A Collection represents a group of objects, known as its elements.

Abstract Data Types

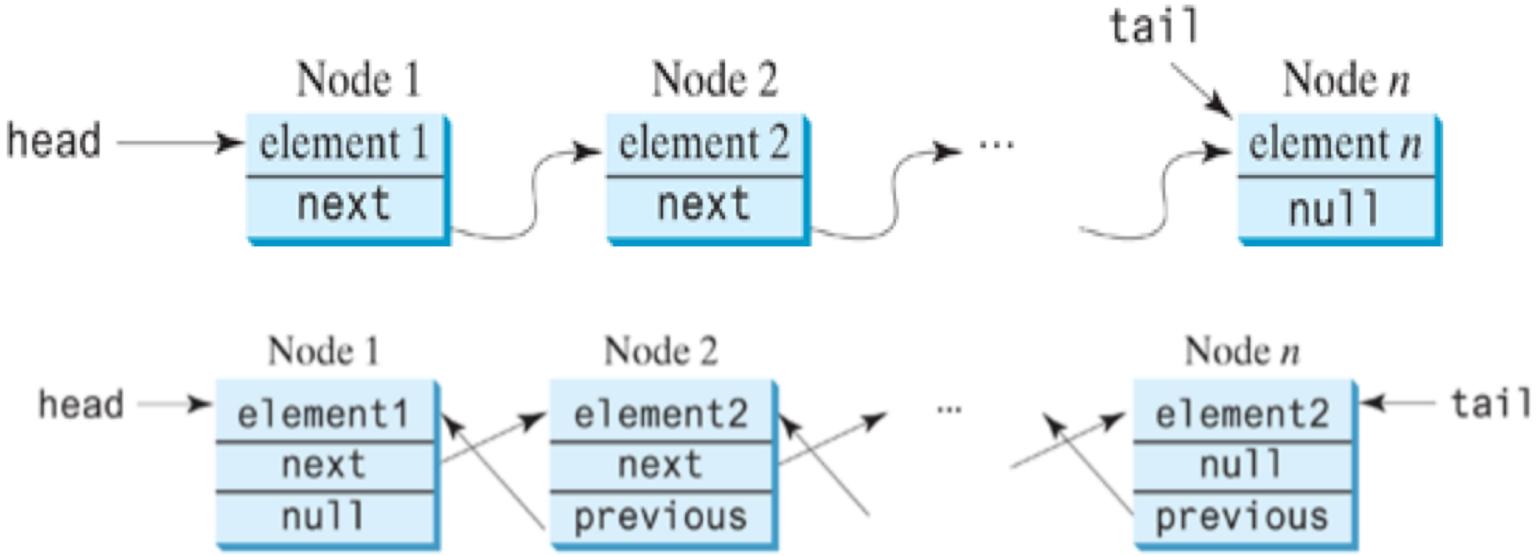


Sets

- A set is an efficient data structure for storing and processing non-duplicate elements.
- Operations:
  - add
  - addAll - adds one set to another
  - remove
  - contains
  - isEmpty
  - size

Linked Lists

- A linked list contains *Nodes* that hold a reference to the next item in the list.
-



LinkedList vs ArrayList

ArrayList:

- Stores elements in an array!
- Every element has an index- Access is fast
- Adding can be expensive if array is full

LinkedList:

- Elements are stored in a sort of chain of “Nodes”
- Nodes have data and a next element
    - Sometimes a previous as well (DoublyLinkedList)
- Access can be slow
- Adding is quick

LinkedList vs ArrayList

Which is more efficient for:

- Adding an element to the beginning?
- Deleting an element in the middle?
- Accessing the last element?
- Accessing an element in the middle?

LinkedList vs ArrayList

Which is more efficient for:

- Adding an element to the beginning? LinkedList
- Deleting an element in the middle?
- Accessing the last element?
- Accessing an element in the middle?

LinkedList vs ArrayList

Which is more efficient for:

- Adding an element to the beginning?
- Deleting an element in the middle? LinkedList
- Accessing the last element?
- Accessing an element in the middle?

LinkedList vs ArrayList

Which is more efficient for:

- Adding an element to the beginning?
- Deleting an element in the middle?
- Accessing the last element? ArrayList or DoublyLinkedList
- Accessing an element in the middle?

LinkedList vs ArrayList

Which is more efficient for:

- Adding an element to the beginning?
- Deleting an element in the middle?
- Accessing the last element?
- Accessing an element in the middle? ArrayList

Kahoot

- https://play.kahoot.it/v2/lobby?quizId=f6733bfb-b581-4baa-94df-6d4dba89dcdf

Worksheet