CSSE 220 Day 29

Exam Review
Generics

Checkout *Generics* project from SVN

Questions

Project demo/presentation Thursday

- Business casual
- Think of it as an internal company presentation, not a presentation to the public
- Five-minute presentation, two minutes for questions, two minutes for transition to next team
- Order of teams will be randomly determined

Project demo/presentation Thursday

- Do a *quick* demo of your project
  - Show off any "extra" features or things that work well
- What part was your team's biggest challenge?
- Show one or two interesting code snippets
  - Highlight your good OO design
- Ask for questions
  - **And ask questions of other teams**

- Before Thursday, practice getting your computer working with a New Olin projector
  - Remember maximum resulution

Final Exam

- Exam is Monday, Feb 18 at 6:00 pm
- Same general format as previous exams
- Same resources:
  ◦ Paper part: 1 page of notes
  ◦ Computer part: Open book, notes, computer; course web pages and ANGEL pages, JDK documentation, programs in YOUR CSSE220 repositories
- Comprehensive, but focused on Ch 9–18
- May include problems that make sure you understand your team's project code

Final Exam – possible topics

- Simple recursion
- Mutual recursion
- Time-space trade-offs
- Basic search algorithms
  - Binary search, linear search
  - Efficiency, best/worst case inputs
- Big-oh notation, estimating big-oh behavior of code

- File I/O, exception handling
- Function objects
- Linked-list implementation
- Basic data structure use and efficiency
  - ArrayList, LinkedList, Stack, Queue, HashSet, TreeSet, HashMap, TreeMap
- Multithreading (not locks)

Final Exam – possible topics

- Interfaces, polymorphism, inheritance and abstract classes
- Exception handling (try, catch, finally, throw, throws)
- OO design and UML class diagrams
- Basic sorting algorithm
  - Insertion sort
  - Selection sort
  - Merge sort
  - Big-oh analysis of each
- Generic programming
- Event handling, layout managers, exploring the Swing documentation
- Your LodeRunner implementation

Generic Types

>> Another way to make code more re-useful

Before Generics...

- Java Collections just stored *Object*s
  - This was better than creating different collection classes for each kind of object to be stored
  - Could put anything in them because of **polymorphism**

- Used class casts to get the types right:
  - ```
    ArrayList songs = new ArrayList();
    songs.add(new Song("Dawn Chorus", "Modern English"));
    …
    Song s = (Song) songs.get(1);
    ```
  - ```
    songs.add(new Artist("A Flock of Seagulls"));
    Song t = (Song) songs.get(2);
    ```

run-time error

Q1

With Generics...

- Can define collections and other classes using type parameters

  ```
  ArrayList<Song> songs = new ArrayList<Song>();
  songs.add(new Song("Dawn Chorus", "Modern English"));
  …
  Song s = songs.get(1); // no cast needed
  songs.add(new Artist("A Flock of Seagulls"));
  ```

  compile-time error

- Lets us use these classes:
  - in a variety of circumstances
  - with strong type checking
  - without having to write lots of casts

Q2

Example

- Create a doubly linked list

- Include *min()* and *max()* methods

- Use **polymorphism** rather than null checks for the start and end of the list

- Include *fromArray()* factory method

Q3-Q5

Generics Recap

▸ Type parameters:
  ◦ **class DLList\<E\>**

▸ Bounds:
  ◦ **class DLList\<E extends Comparable\>**
  ◦ **class DLList\<E extends Comparable\<E\>\>**
  ◦ **class DLList\<E extends Comparable\<? super E\>\>**

▸ Generic methods:
  ◦ **public static \<T\> void shuffle(T[ ] array)**

http://docs.oracle.com/javase/tutorial/java/generics/index.html

Q6-7, turn in

LodeRunner Work Time