PROJECT 3 INSTRUCTIONS

You're running a kitten shelter now and trying to get your kitties adopted! And everyone knows that the cuteness of a kitten is directly correlated to its adoptability. Your task in this Project is to build a Priority Queue that will always have the CUTEST kitten as its first element.

Your Priority Queue will be implemented with a linked list instead of an array, and it will make use of a stack (also a linked list) in order to update and resort every Kitten as necessary.

Here's a breakdown of the classes you'll be working with in this Project. Asterisks around the class name mean you have to do some work to fully implement it, and instructions for these implementations are in the comments for each class:

Kitten: A new private int has been added: cuteness. This field is updated whenever the kitten eats or plays; it is inversely tied to hunger, so that when a kitten is less hungry and playful, it gets cuter, but when it's hungry and not playful, it gets less cute.

Fields: int hunger, int cuteness, boolean playful, String name
Methods: changeState(States state), eat(int food), play(), meow(), checkPlay(), getName(), getHunger(), getCuteness()

KittenLink: A "link" for use in KittenLL (a linked list). You don't need to add anything to this class; you just need to call its methods in order to implement a KittenLL. An additional int, cuteLvl, is simply the current cuteness of the Kitten stored in this link, used for quick comparison and indexing instead of calling the Kitten getCutness() method every time.

Fields: public Kitten k, public int cuteLvl, public KittenLink next
Methods: display()

***** KittenLL ***:** A linked list of Kitten objects. You will need to implement this using the KittenLink class above. This is a singly-linked list. Since we're using this as both a stack and a queue in other classes, and we'll implement a special insert method in the queue class to make it a priority queue, we only need these methods here.

Fields: public KittenLink first
Methods: boolean isEmpty(), insertFirst(Kitten k), deleteFirst(), displayList()

***** KittenStack ***:** A stack of Kitten objects implemented as a KittenLL. Whenever it's time to feed the Kittens, the KittenAdopter will pop every Kitten from the KittenPQ (see below), feed it to update its cuteness, and push it onto this stack. The Kittens will then be popped from this stack back into KittenPQ, which should automatically insert each one in the list according to its new cuteness level. A helper method, walk(), is provided to check whether insertions are correctly done.

Fields: private KittenLL stack
Methods: boolean isEmpty(), push(Kitten k), Kitten pop(), Kitten peek(), int walk(), display()

***** KittenPQ ***:** A priority queue of Kitten objects sorted by cuteness. This is the "roster" we'll be using to track the kittens in our shelter. The Kitten objects are stored in KittenLinks in a KittenLL. Since this is a priority queue, whenever a new Kitten is inserted, this class will find its proper place in the queue based on its cuteness. A helper method, walk(), is provided to check whether insertions are correctly done.

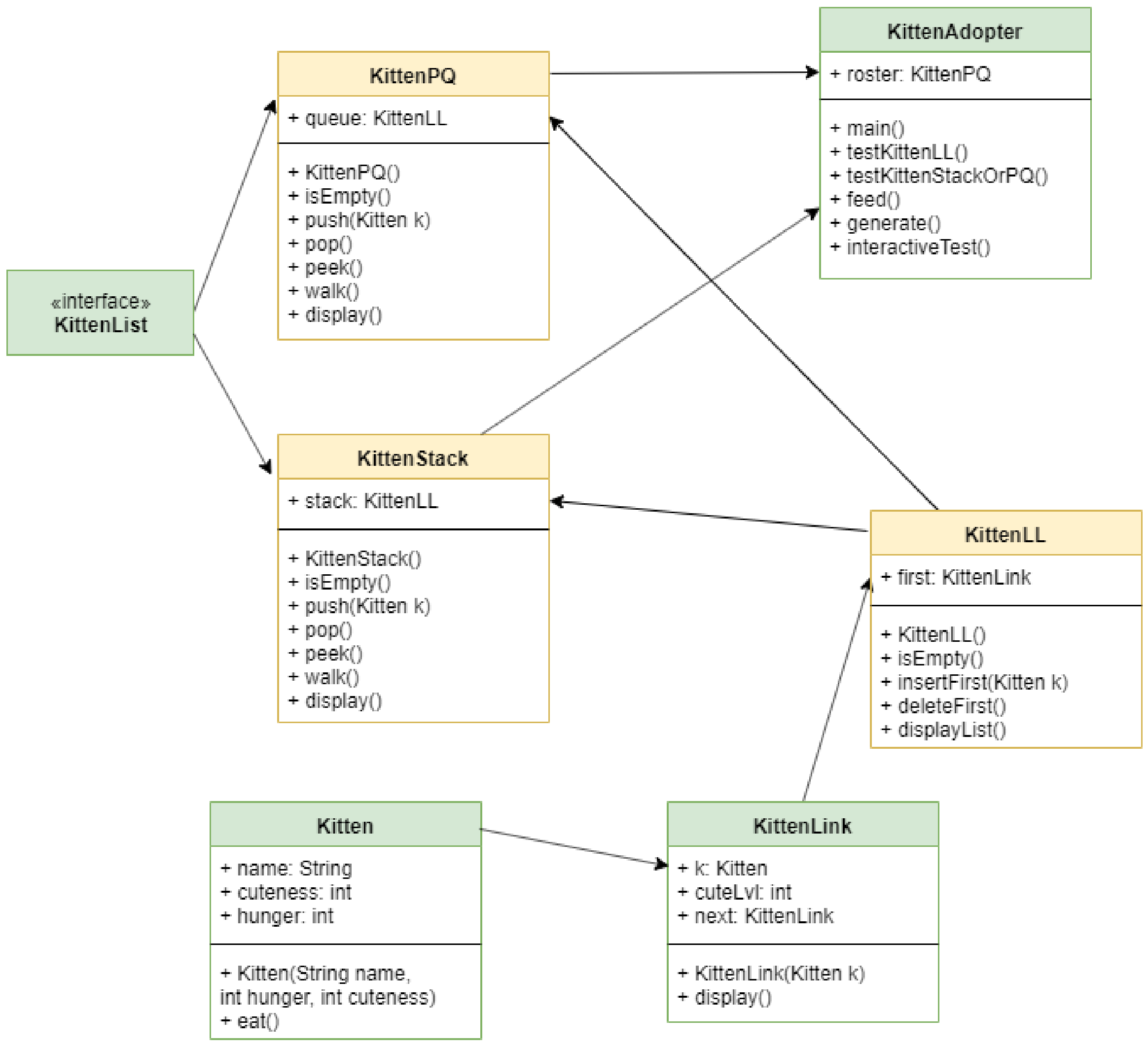
Fields: private KittenLL queue
Methods: boolean isEmpty(), push(Kitten k), Kitten pop(), Kitten peek(), int walk(), display()

KittenAdopter: The main program that will test your implementation of these data structures. It consists of several standalone test methods for KittenLL, KittenStack, and KittenPQ as well as an interactive test method. Feel free to modify these methods as you require.

Fields: KittenPQ roster
Methods: various (no implementation required)

There is also a Java Interface, KittenList, which is implemented by KittenStack and KittenPQ. You don't need to do anything with this, but it needs to be in the same directory as the rest of your files.

Remember that you only have to code up parts of three classes: KittenLL, KittenStack, and KittenPQ. Both KittenStack and KittenPQ rely on methods in KittenLL, so start there. I've included a UML diagram showing the way these classes and interface relate below (green titles mean they're complete, yellow that they require you to implement them):



«interface»
KittenList
KittenPQ
+ queue: KittenLL
+ KittenPQ()
+ isEmpty()
+ push(Kitten k)
+ pop()
+ peek()
+ walk()
+ display()
KittenAdopter
+ roster: KittenPQ
+ main()
+ testKittenLL()
+ testKittenStackOrPQ()
+ feed()
+ generate()
+ interactiveTest()
KittenStack
+ stack: KittenLL
+ KittenStack()
+ isEmpty()
+ push(Kitten k)
+ pop()
+ peek()
+ walk()
+ display()
KittenLL
+ first: KittenLink
+ KittenLL()
+ isEmpty()
+ insertFirst(Kitten k)
+ deleteFirst()
+ displayList()
Kitten
+ name: String
+ cuteness: int
+ hunger: int
+ Kitten(String name, int hunger, int cuteness)
+ eat()
KittenLink
+ k: Kitten
+ cuteLvl: int
+ next: KittenLink
+ KittenLink(Kitten k)
+ display()