CSE12 Discussion 2

Created by:
Pooja Bhat & Maxim Jiao

Quia Review Quiz

Which of the following is the most analogous to an ADT (Abstract Data Type)

A. Data Structure
B. Abstract Programming Interface (API)
C. Array
D. Java Generics
E. Iclicker

Quia Review Quiz

Which of the following is the most analogous to an ADT (Abstract Data Type)

A. Data Structure
B. Abstract Programming Interface (API)
C. Array
D. Java Generics
E. Iclicker

Answer: B

What is an ADT?

- A mathematical model (defined by behaviors from a user's point of view)

An API is like an ADT that is language specific

The Data Structure:

- defines how the data is stored / how the ADT model is implemented (linked structure, contiguous array)
- "Developer's point of view" internals of an ADT

Speed of Operations

- Consider an Arraylist. Internally it is not full, and the number of elements inserted so far is known. The elements are not sorted.

  Choose the operations listed below that are fast regardless of the number of elements contained in the ArrayList. (In other words, takes only several instructions to implement).

  - Insertion
  - Insertion at a given index
  - Deletion at a given index
  - Searching for a specific element
  - Finding the maximum value (unsorted)
  - Getting data from a specified index
  - Replacing an element at a specified index

Speed of Operations

- Consider an Arraylist. Internally it is not full, and the number of elements inserted so far is known. The elements are not sorted.

  Choose the operations listed below that are fast regardless of the number of elements contained in the ArrayList. (In other words, takes only several instructions to implement).

  - Insertion
  - Insertion at a given index
  - Deletion at a given index
  - Searching for a specific element
  - Finding the maximum value (unsorted)
  - Getting data from a specified index
  - Replacing an element at a specified index

Generics

```
public class Test<E>{

E element;

...

}
```

How can we Determine what E is?

A. Can’t tell from just this

B. E is generic, so you have to treat it like Object, using downcasting

C. It will be specified via each method’s return type

D. Using generics this way is terrible coding practice

Generics

```
public class Test<E>{

E element;

...

}
```

When you create a new object of type Test, you would make it:

```
Test<Object> var = new Test<>();
```

The Object between the <> lets the user specify what type E is

(Integer, String, etc)

How can we Determine what E is (what kind of object)?

A. Can't tell from just this

B. E is generic, so you have to treat it like Object, using downcasting

C. It will be specified via each method's return type

D. Using generics this way is terrible coding practice

More Generics:

which of the following best describes the code below (assuming imports are done correctly

```
public static void main (String[] args) {
  List<int> myArray = new ArrayList<int>(4);
  int index = 0;

  while() {
    if(index > 4)
      break;
    myarray.add(index++,index);
  }
}
```

A. The loop will run forever
B. Compile error because myarray was improperly initialized
C. myarray will end up as [0,1,2,3]
D. myarray will end up as [1,2,3,4]
E. IndexOutOfBounds error before the while loop exits

More Generics:

which of the following best describes the code below (assuming imports are done correctly

```
public static void main (String[] args) {
  List<int> myArray = new ArrayList<int>(4);
  int index = 0;

  while() {
    if(index > 4)
      break;
    myarray.add(index++,index);
  }
}
```

A. The loop will run forever

B. Compile error because myarray was improperly initialized

C. myarray will end up as [0,1,2,3]

D. myarray will end up as [1,2,3,4]

E. IndexOutOfBounds error before the while loop exits

Why?

More Generics:

which of the following best describes the code below (assuming imports are done correctly

```
public static void main (String[] args) {
  List<int> myArray = new ArrayList<int>(4);
  int index = 0;

  while() {
    if(index > 4)
      break;
    myarray.add(index++,index);
  }
}
```

A. The loop will run forever

B. Compile error because myarray was improperly initialized

C. myarray will end up as [0,1,2,3]

D. myarray will end up as [1,2,3,4]

E. IndexOutOfBounds error before the while loop exits

Why?

Generics can only take Objects, int is a primitive!

More Generics:

What about now?

which of the following best describes the code below (assuming imports are done correctly

```
public static void main (String[] args) {
  List<Integer> myArray = new ArrayList<>(4);
  int index = 0;

  while() {
    if(index > 4)
      break;
    myarray.add(index++,index);
  }
}
```

A. The loop will run forever
B. Compile error because myarray was improperly initialized
C. myarray will end up as [0,1,2,3]
D. myarray will end up as [1,2,3,4]
E. IndexOutOfBounds error before the while loop exits

More Generics:

What about now?

which of the following best describes the code below (assuming imports are done correctly

```
public static void main (String[] args) {
  List<Integer> myArray = new ArrayList<>(4);
  int index = 0;

  while() {
    if(index > 4)
      break;
    myarray.add(index++,index);
    //this is the same as add(index, index+1)
  }
}
```

A. The loop will run forever

B. Compile error because myarray was improperly initialized

C. myarray will end up as [0,1,2,3]

D. myarray will end up as [1,2,3,4]

E. IndexOutOfBounds error before the while loop exits

Interfaces:

Assuming that the class (ThisClass) that implements an interface (ThisInterface), and both are written correctly.
ThisClass has a default constructor that takes no arguments.

Which of the following will compile?

- ThisInterface x;
- ThisInterface x = new ThisInterface();
- ThisInterface x = new ThisClass();
- ThisClass x;
- ThisClass x = new ThisInterface();

Assuming that the class (ThisClass) that implements an interface (ThisInterface), and both are written correctly.
ThisClass has a default constructor that takes no arguments.

Which of the following will compile?

- ThisInterface x;
- ThisInterface x = new ThisInterface();
- ThisInterface x = new ThisClass();
- ThisClass x;
- ThisClass x = new ThisInterface();

You can't instantiate an interface, but you can declare it as a type, as long as the actual class implements that interface!

LinkedLists (LL)

- Data structure which consisting of a list of nodes
- What are nodes?
  - Think of LinkedLists as a train
  - The engine of the train is the “front” or the “head”
  - After that, each car contains things inside of it, like people or goods (data)
  - Each car has a door leading to the “next” car, which is just like it but with a different “next” car and different “data”
  - Of course, you can go to the “previous” car as well
- What other fields should LinkedLists have?
  - Such as number of elements
  - Where the front of the list is
  - What about the end of the list?

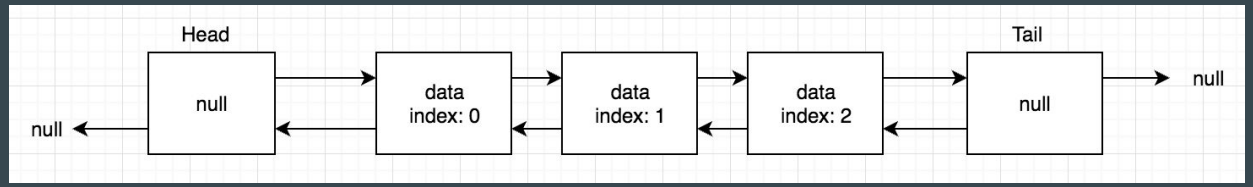
Dummy nodes

A dummy node is a node that has null value i.e. does not carry any data. Dummy nodes are often used as sentinel nodes in a list.

Dummy nodes are used in linked data structures to avoid special cases with the start or end of the data structure. The dummy head has no value, but it does have a link that can be followed to get to the "real" head of the list.

Each link is just a next or prev that points to the corresponding node

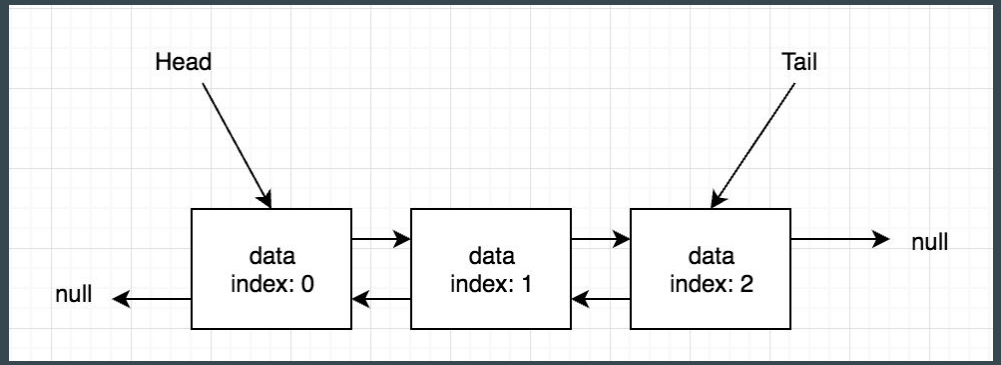
Consider a add(data) method in a list a)with dummy nodes b) without dummy nodes



Do we Have to use a Dummy Node?

- For this assignment, yes, but in general, no

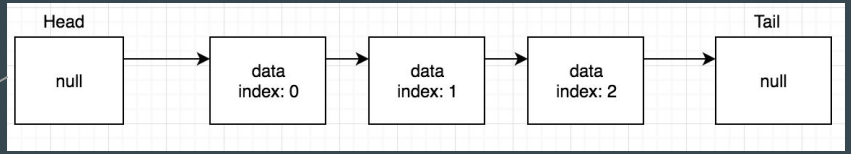
  How would a Doubly Linked List with no dummy nodes look?

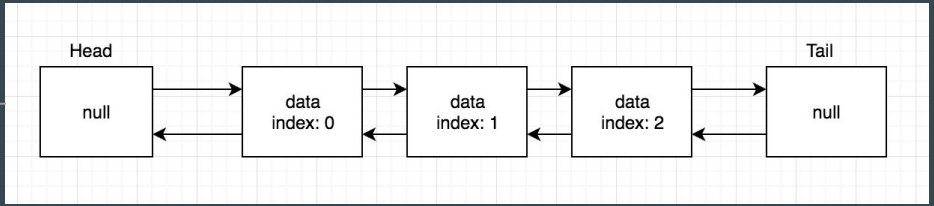


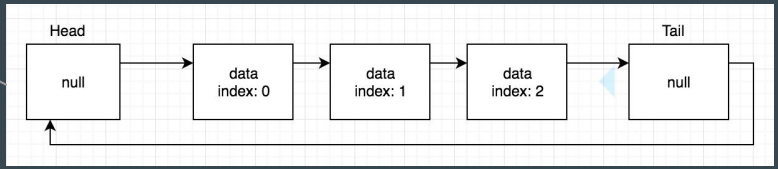
The Head and Tail are the actual first and last nodes!

Types of LinkedLists

- Singly Linked List
- Doubly Linked List
- Circular Linked List







Each of these could either have a head or both head and tail.

Why LinkedLists? Where do we use them?

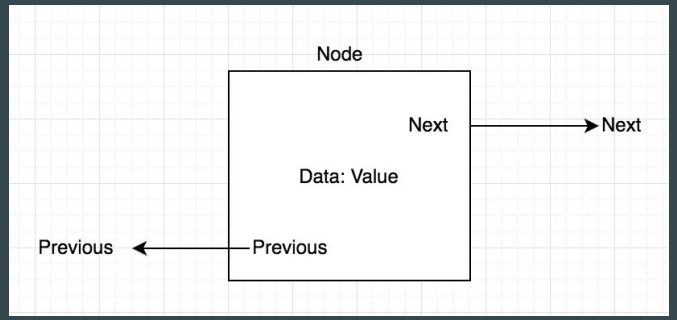
- Microsoft Image Viewer: In the UI, '->' button is your node->next; The '<-' button is your node->prev.

- The browser cache which allows you to hit the BACK button (a linked list of URLs)

More Tips for HW2

- There are a LOT of methods that you have to write, but each one usually doesn't require a lot of code to complete
  - Once you're done, think about the structure and design that went into making a linked list (it could be pretty insightful for future homework assignments and personal projects

- Once you write the Node class, you're just building a Linked List by linking these Nodes



HW2 - Part 1 and Part 2

- LinkedListTester contains tests against Java's LinkedList class
- You can add to it to test the different methods of the LinkedList such as:
  - get(index) //throws IndexOutOfBoundsException
  - add(data)
  - add(index,data) //throws ?
  - set(index,data) //returns? Throws?
  - remove(index) //returns? Throws?
  - Etc
- In Part 2, you would develop your own implementation of the above methods, but for a DoublyLinkedList.
- HINT: You can make use of the private method getNth for a lot of the above methods.
- TIP: Do not forget to update the instance variable nelems as and when necessary

Example:
How would we approach the add(int index, E element) method?

1. Get to the Node whose current index is specified
2. Create a new Node to be inserted!
3. Link the new Node so that it fits into the list
4. Unlink the Nodes where we want to add the new one

- ORDER YOU DO THESE DOES MATTER!!!

Exceptions

```
//How to throw an exception
int somefunc() throws ThisIsAnException {
  if (SOME_CONDITION)
       throw new ThisIsAnException("MESSAGE");
  //code...
}

//How to catch an exception
try {
    int res = somefunc()
    //did not throw an exception. Normal behavior or not?
    //code..
} catch(ThisIsAnException e) {
     //exception caught. Normal behavior or not?
}
```