7.2 LinkedList

LinkedList

The **LinkedList** is very similar in syntax to the **ArrayList**.

You can easily change an ArrayList to a LinkedList by changing the object type.

```
import java.util.LinkedList;

public class MyClass {
 public static void main(String[ ] args) {
   LinkedList<String> c = new LinkedList<String>();
   c.add("Red");
   c.add("Blue");
   c.add("Green");
   c.add("Orange");
   c.remove("Green");
   System.out.println(c);
 }
}
// Outputs [Red, Blue, Orange]
```

Note:

You cannot specify an initial capacity for the **LinkedList**.

Q: Fill in the blanks to create a LinkedList, add "Hey", and print it.

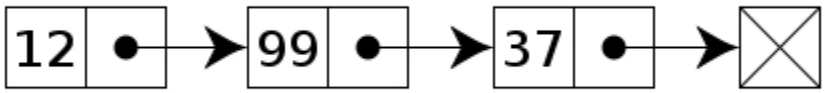
```
_____<String> list =
    new LinkedList_____();
list.add(_____);
System.out.println(list.get(_____));
```

LinkedList vs. ArrayList

The most notable difference between the **LinkedList** and the **ArrayList** is in the way they store objects.

The ArrayList is better for **storing** and **accessing** data, as it is very similar to a normal array. The LinkedList is better for **manipulating** data, such as making numerous inserts and deletes.

In addition to storing the object, the LinkedList stores the memory address (or link) of the element that follows it. It's called a LinkedList because each element contains a link to the neighboring element.



You can use the enhanced for loop to iterate over its elements.

```
LinkedList<String> c = new LinkedList<String>();
c.add("Red");
c.add("Blue");
c.add("Green");
c.add("Orange");
c.remove("Green");

for(String s: c) {
 System.out.println(s);
}
/* Output:
Red
Blue
Orange
*/
```

Note:

- Use an **ArrayList** when you need rapid access to your data.

- Use a **LinkedList** when you need to make a large number of inserts and/or deletes.

Q: For a program with large number of inserts and deletes, it is better to use the…

- ArrayList
- LinkedList