Java Collections

- java.util
- ArrayList
- LinkedList

See Also

https://docs.oracle.com/javase/tutorial/collections/

https://docs.oracle.com/javase/tutorial/collections/intro/

http://www.javatpoint.com/collections-in-java

The “java.util” Collections

- The “collections” are in java.util

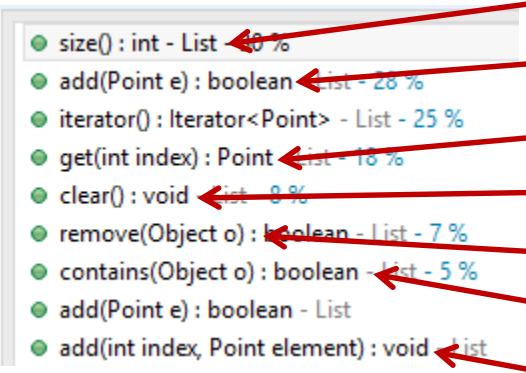
- List<?> access by numerical index
- Set<?> only unique objects (no indexing)
- Map<?,?> association between key and value

- Static methods in the class Collections
  - Sorting
  - Searching

List of Objects

- List<?> is the base of several implementations

```
List<Point> points = null;

points.
```



- Number of objects in the collection
- Add object to the end
- Get object at index
- Remove everything
- Remove a specific object
- Search for object
- Insert object AT index

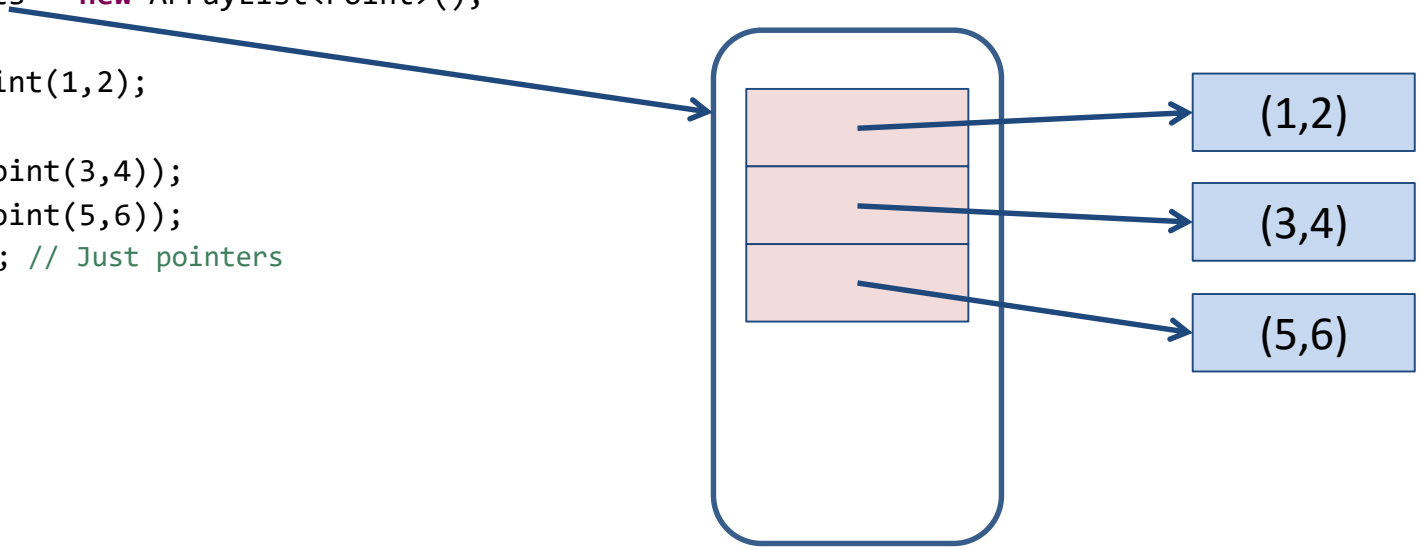
Based on an Array

- Similar to what we made
- Array of POINTERS

```
ArrayList<Point> points = new ArrayList<Point>();
List<Point> points = new ArrayList<Point>();

Point p = new Point(1,2);
points.add(p);
points.add(new Point(3,4));
points.add(new Point(5,6));
points.add(1,null); // Just pointers
points.add(0,p); // Duplicate pointers
points.remove(3);
points.add(0,p);
points.set(2, new Point(7,8));

System.out.println(points.size());
System.out.println(points);
```

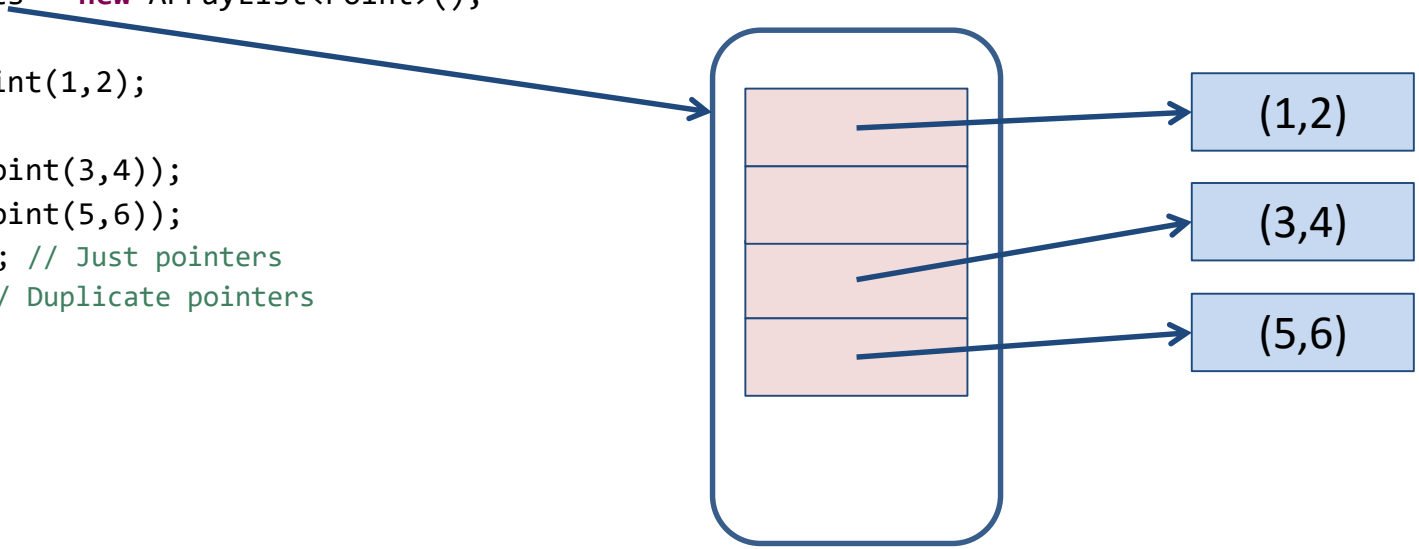
Based on an Array

```
List<Point> points = new ArrayList<Point>();

Point p = new Point(1,2);
points.add(p);
points.add(new Point(3,4));
points.add(new Point(5,6));
points.add(1,null); // Just pointers
```



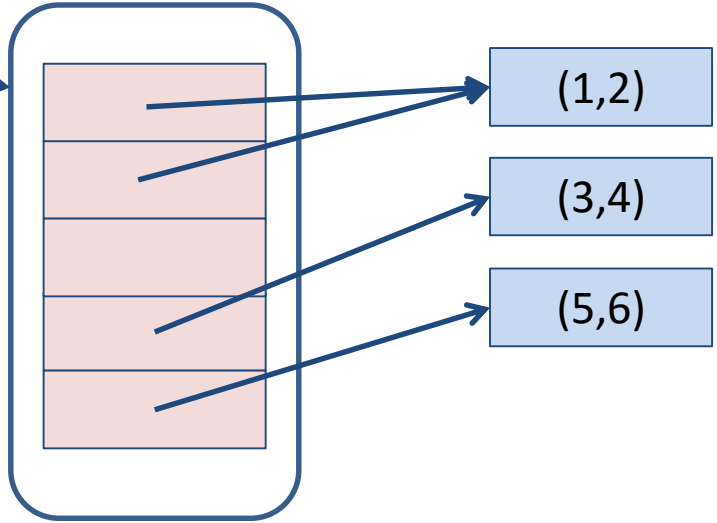
Based on an Array

```
List<Point> points = new ArrayList<Point>();

Point p = new Point(1,2);
points.add(p);
points.add(new Point(3,4));
points.add(new Point(5,6));
points.add(1,null); // Just pointers
points.add(0,p); // Duplicate pointers
```



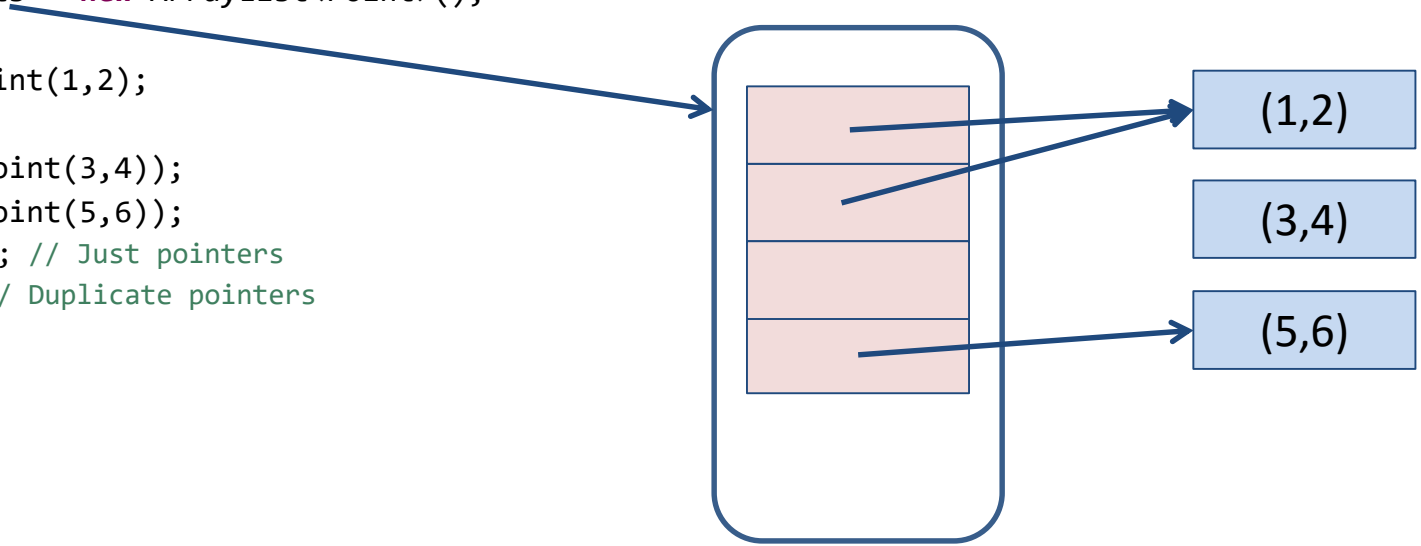
Based on an Array

```
List<Point> points = new ArrayList<Point>();

Point p = new Point(1,2);
points.add(p);
points.add(new Point(3,4));
points.add(new Point(5,6));
points.add(1,null); // Just pointers
points.add(0,p); // Duplicate pointers
points.remove(3);
```



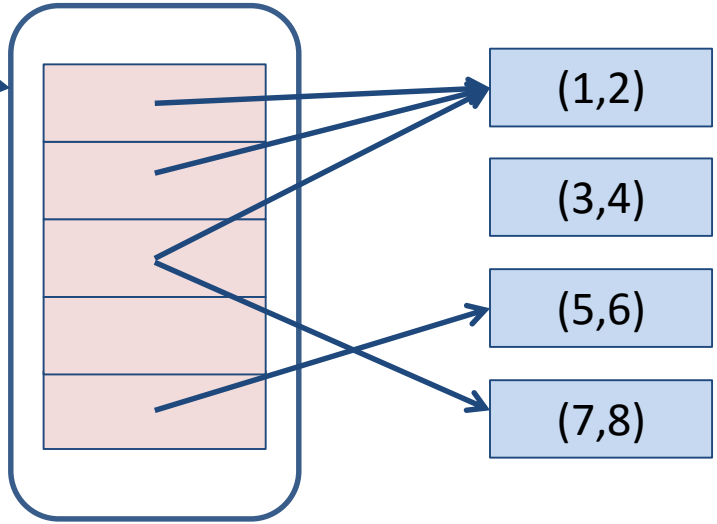
Based on an Array

```
List<Point> points = new ArrayList<Point>();

Point p = new Point(1,2);
points.add(p);
points.add(new Point(3,4));
points.add(new Point(5,6));
points.add(1,null); // Just pointers
points.add(0,p); // Duplicate pointers
points.remove(3);
points.add(0,p);
```



Based on an Array

```
List<Point> points = new ArrayList<Point>();

Point p = new Point(1,2);
points.add(p);
points.add(new Point(3,4));
points.add(new Point(5,6));
points.add(1,null); // Just pointers
points.add(0,p); // Duplicate pointers
points.remove(3);
points.add(0,p);
points.set(2, new Point(7,8));

System.out.println(points.size());
System.out.println(points);
```



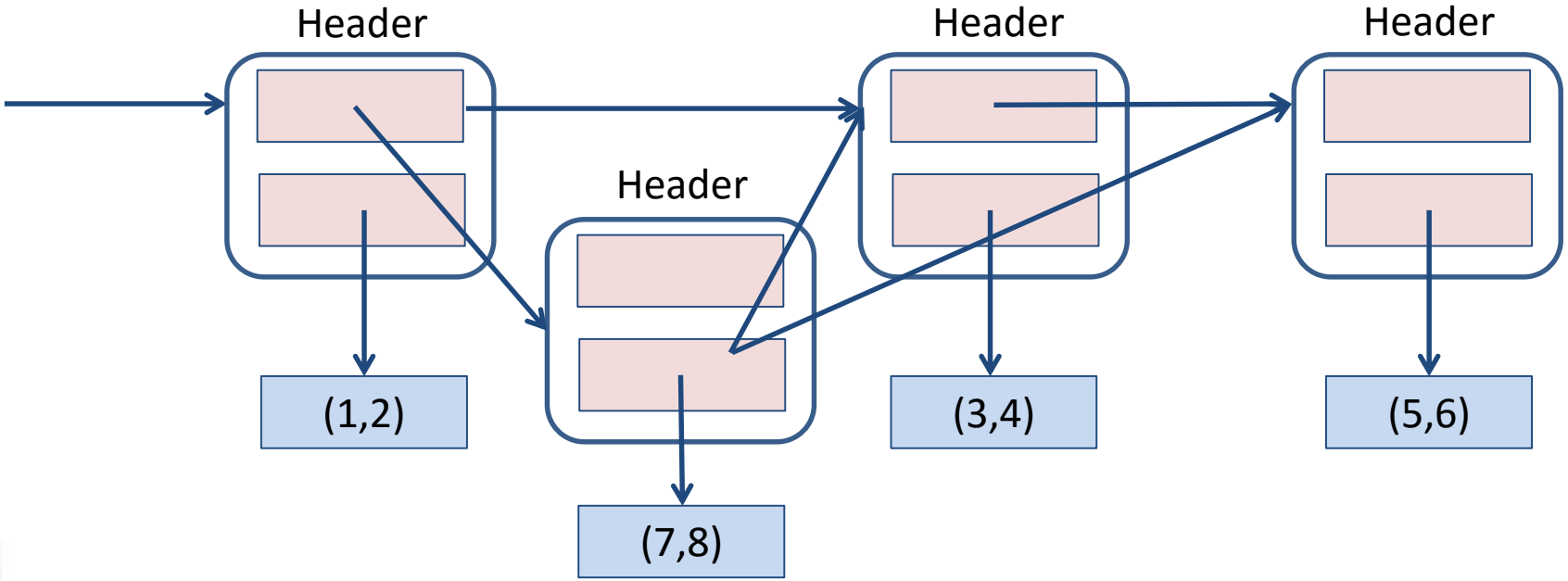

```
5
[1,2, 1,2, 7,8, null, 5,6]
```

ArrayList Properties

- Random access (gets and sets) are very fast since they are just array lookups
- Adds and deletes are expensive because you have to shuffle lots of memory around
- When you need do mostly lookups and few changes, ArrayList is perfect.

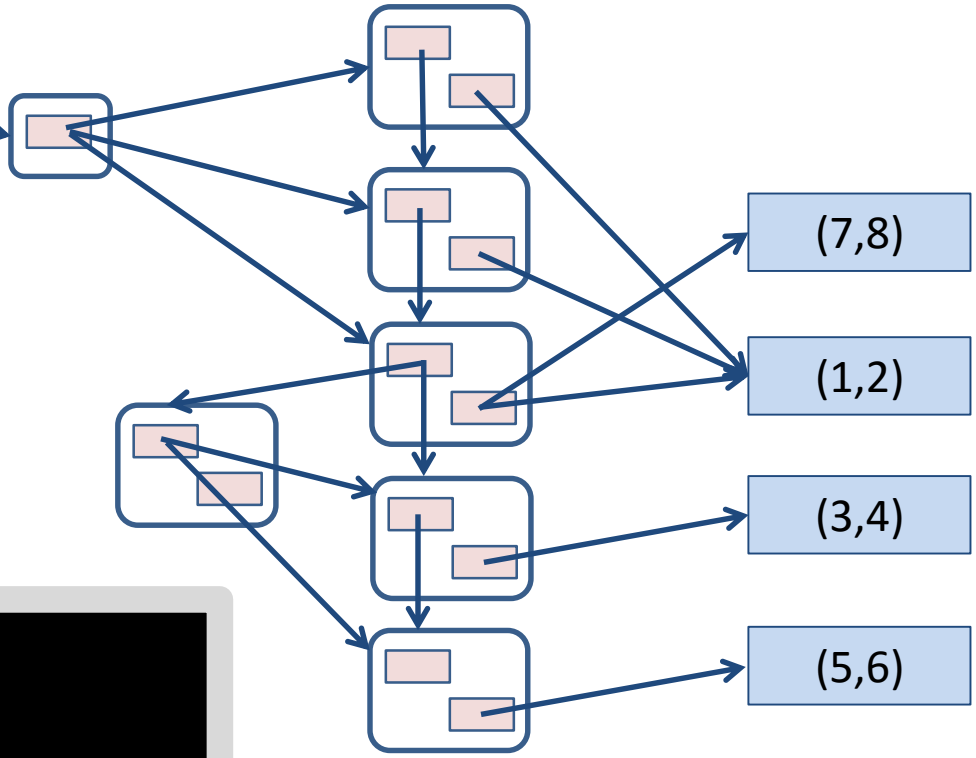
Linked List

- Header objects point to one another in a chain
- The header also points to the "payload" object
- The header objects don't "move" like in ArrayList



Same Interface

```
List<Point> points = new LinkedList<Point>();

Point p = new Point(1,2);
points.add(p);
points.add(new Point(3,4));
points.add(new Point(5,6));
points.add(1,null); // Just pointers
points.add(0,p); // Duplicate pointers
points.remove(3);
points.add(0,p);
points.set(2, new Point(7,8));

System.out.println(points.size());
System.out.println(points);
```




```
5
[1,2, 1,2, 7,8, null, 5,6]
```

LinkedList Properties

- Random access (gets and sets) is very slow since you must follow the pointers down the chain
- Adds and deletes are quick because you only change pointers … not shuffle memory
- When you need do inserts and deletes with few random access lookups, LinkedList is perfect.

Tinkering

- Create some points and add them to a List: first ArrayList then change to a LinkedList.
- Use the built-in toString to print your lists.