

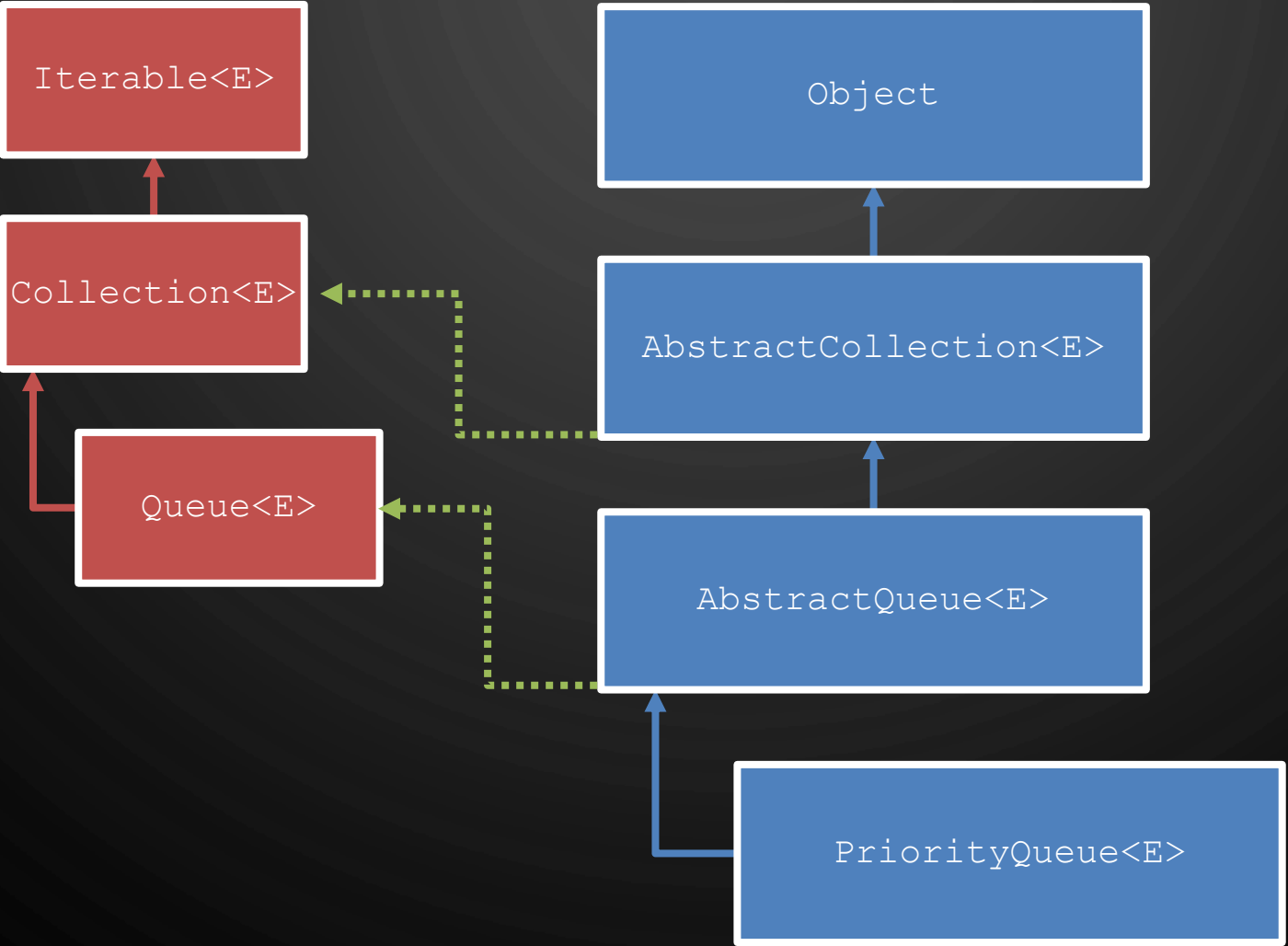


# JAVA PRIORITY QUEUE

# SUMMARY OF CLASSES (PRIORITY QUEUE RELATED)

- [PriorityQueue<E>](#) - array-based heap implementation of minimum priority queue
- [Comparator<E>](#) - can be useful for defining your own comparison between objects
- Others outside the scope of this course
- To find how to use them, go to the Java API!

Interfaces  
Classes



# EXAMPLE OF USING PRIORITYQUEUE<E>

```
1. Scanner s = new Scanner(new File("numbers.txt"));
```

```
2. PriorityQueue<Integer> numbers = new PriorityQueue<>();
```

```
3. while (s.hasNextInt())
```

```
4.     numbers.add(s.nextInt());
```

```
5. ...elsewhere...
```

```
6. int sum = 0;
```

```
7. while (!numbers.isEmpty())
```

```
8.     sum += numbers.poll(); //poll is removeMin()
```

# COMPARISON IN JAVA

- It is not a Boolean less than. It is always an integer value  $i$ . So for two objects  $a$  and  $b$ , a comparison  $comp(a, b)$  returns:
  - $i < 0$  implies  $a$  is ordered before  $b$ , e.g.,  $<$
  - $i = 0$  implies  $a.equals(b)$
  - $i > 0$  implies  $a$  is ordered after  $b$ , e.g.,  $>$
- First method - no new class. Have your class **E** implement Comparable<E>, which requires the **int** `compareTo (E o)` method
- Second method – separate comparator class that implements **Comparator<E>** interface
  - Must define `compare (E o1, E o2)` and `equals (Object o)`
    - Here equals is a comparison to another comparator

# COMPARISON IN JAVA – FIRST METHOD

## IMPLEMENT COMPARABLE

```
public class Foo implements Comparable<Foo> {  
    private int x;  
    public Foo(int x) {this.x = x;}  
    // ... stuff ...  
    public int compareTo(Foo other) {  
        return x - other.x; // Note, <0 means this is before other;  
                            // = 0 means this.equals(other);  
                            // and >0 means this is after other  
    }  
}
```

# COMPARISON IN JAVA – SECOND METHOD

## CREATE SEPARATE COMPARATOR

```
import java.util.Comparator;

public class CompareFoo implements Comparator<Foo> {
    public int compare(Foo a, Foo b) {
        return a.getX() - b.getX(); // Assume Foo does not implement
                                     // Comparable, and has a public getX()
                                     // accessor
    }

    public boolean equals(Object obj) {
        return obj instanceof CompareFoo;
    }
}
```

# PROBLEM

## EVENT DRIVEN SIMULATION

- Event driven simulation – you want to estimate the profit for a coffee shop. There is an input file online stating the number of seats in the shop, the price per cup of coffee, and arrive events with a given time (integer) and number of partisans (integer) (1 pair per line)
- Use a priority queue of events, ordered by time to see how much profit the store will earn over this period. Rules:
  - Arrive event - If a group enters and there are not enough seats they will leave. If they stay, an order event will be created at the current time + 1 + a random number below 4
  - Order events - Every partisan of the group will buy 1 or 2 cups of coffee. Each orderEvent will also spawn a leaveEvent at the currentTime + 1 + a random number below 10.
  - Leave event – When a group leaves, their chairs are opened up to another group
- Create an object oriented solution to this problem with your team. PLAN-IMPLEMENT-TEST!





# EVENT DRIVEN SIMULATION

## MAIN LOOP ALGORITHM

1. Priority queue of Events  $PQ$
2. **while**  $\neg PQ.is\text{Empty}()$  **do**
3.  $PQ.remove\text{Min}().process()$

# EVENT DRIVEN SIMULATION

## POSSIBLE CLASS HIERARCHY

