Java Interfaces:
Consider the Bag API, page 121

```java
public class Bag<Item> implements Iterable<Item> {
    Bag() { /* constructor, not part of Java interface */
    void add(Item item);
    boolean isEmpty();
    int size();
}
```

Here is an interface with all the methods of the API:

```java
public interface Bag<Item> implements Iterable<Item> {
    void add(Item item);
    boolean isEmpty();
    int size();
}
```

Java Interfaces can describe a subset of the methods of a class

```java
public class Bag<Item> implements Iterable<Item> {
    Bag() { /* constructor, not part of Java interface */
    void add(Item item);
    boolean isEmpty();
    int size();
}
```

Interface with some of the methods of the Bag API:

```java
public interface Bag<Item> implements Iterable<Item> { 
    void add(Item item);
    boolean isEmpty();
}
```

• Note that a class needs to implement all the methods of the interface to have "implements Bag<Item>" in its class definition, so more classes will qualify if the interface is simpler.

Priority queues

• A priority queue holds elements that have a “priority” value, and we can new elements, or get back out the one with minimum priority value.

- Insert
- DeleteMin

Priorities queues

• For example, suppose incoming messages to an email support center each have a priority (lowest number is highest priority.)
• Unanswered messages are held in a priority queue, and new messages are added to it.
• When a support engineer becomes free, they take out the message of highest priority to work on.
• It might be a recently received message or an old message—this is not a normally-ordered queue.
Operations of a Priority Queue: APIs

<table>
<thead>
<tr>
<th>Classic Setup</th>
<th>S&amp;W MinPQ&lt;Key&gt; (p. 309)</th>
<th>JDK PriorityQueue&lt;Key&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>insert(Key item)</td>
<td>add(Key item), offer(Key item), same</td>
</tr>
<tr>
<td>FindMin</td>
<td>Key min()</td>
<td>Key peek()</td>
</tr>
<tr>
<td>DeleteMin</td>
<td>Key delMin()</td>
<td>Key poll()$</td>
</tr>
</tbody>
</table>

Simple Priority queue example using element’s compareTo

```java
import java.util.PriorityQueue;
public class PriorityQueueDemo {
    public static void main(String[] args) {
        PriorityQueue<Integer> minPQ = new PriorityQueue<>();
        minPQ.add(4);
        minPQ.add(3);
        minPQ.add(5);
        System.out.println("MinPQ: "+minPQ);
    }
}
```

Output:
```
MinPQ:[3, 4, 5]
```

Priority queue example from earlier slides (APICollection)

```java
// Java Comparator method *comparing* builds a specific-field Comparator, here on GPA
Comparator<Student> byGpa = Comparator.comparing(Student::getGpa);
PriorityQueue<Student> pq = new PriorityQueue<Student>(byGpa);
pq.add(new Student("Ann", 100, 2.3));
pq.add(new Student("Ling", 104, 4.0));
pq.add(new Student("Dave", 104, 3.1));
System.out.println(pq); // note duplicates (equal id)
// Use DeleteMin to pull them out in GPA order--
while (!pq.isEmpty()) {
    Student x = pq.poll();
    System.out.println(x);
}
```

Priority queue implementation

- It is possible to implement a PQ with TreeMap.
- We can use the domain set (the map keys) to hold the PQ elements, and the range element (map value) to hold the duplicate count for this key.
- We can supply a Comparator to the TreeMap constructor to order the map by the priority.
- Then insert works as usual, and deleteMin finds the first key in keySet via an iterator, then uses get() for its multiplicity, and either decreases the count by one or removes the key if the count reaches 0.

Event driven simulations

- Consider a time-line, with points representing events.
- Each event fires off future events, adding points further to the right on the line, sometimes near, sometimes further, past other events.
- How do we keep track of which event is next? Answer: Put the events in a priority queue with time = priority.
- Note that a simulation itself takes hardly any time. It processes events in time order, typically handling a whole day in a second or less.
Event-Driven Simulation - examples

- Bank Teller Problem
  - Customers arrive and wait in line until one of the k tellers is available.
  - How long on the average a customer has to wait?
  - What % of the time tellers are actually servicing requests?
  - Applications – determine ideal number of tellers; statistics on average waiting time etc.
- Event-driven simulation
  - Start a simulation clock at zero ticks and advance the clock to the next event time at each stage.
  - The event is serviced and statistics are collected

Event-Driven Simulation: Particle Simulation in S&W

- S&W pp. 856-865
- Circular discs/balls glide across flat horizontal surface, collide with each other and the walls (in a rectangle)
- Starting from one point in time and one ball, we figure out when it will hit a wall or another ball, that collision is an event, add it to PQ, etc.
- Example video
- The simulation itself only calculates the collisions. To make a video like this, we would also have to work out a lot of intermediate points on the straight lines between collisions.

Call Bank Simulation, from Weiss

- Setup: Customer service is accessed by dialing one telephone number
- If any one of the operators (customer service reps) in the bank is available, the user is connected.
- If all the operators are busy, the phone will give a busy signal (simple case)
- The variables are:
  - Number of operators in the bank
  - Probability distribution that governs dial-in attempts
  - Probability distribution that governs connect time
  - How long to run the simulation

Types of events in call bank simulation

1. Customer arrival event
   Check for an available operator. If none, signal busy. If yes, process the customer, compute simulated departure time and add the departure event (hangup) to the set of events waiting to happen.
2. Customer departure event
   Gather statistics for the customer. Make the operator available for another call.

Call Simulator

- One call is coming in every minute, that is, each call event generates another call event one minute later.
- The calls each last a random length of time, with average of 5 minutes.
  - Each call generates a hangup event in the future in x minutes, where x is random with average 5 min.
  - It also generates a next call in 1 min
- How do we keep track of which event is next?
  - Answer: use a priority queue with time = priority.
Sample Trace for the Call Bank

One call comes in every minute.

Call Simulator

- The simulation uses the Poisson distribution, to specify the call duration $x$:

$$P(x) = a^x \cdot e^{-a}/x!$$

Where $a$ is the average

Here $a = \lambda$

CallSim Class

```java
public class CallSim {
    private class Event implements Comparable<Event> {
        //...
    }
    public CallSim(int operators, double avgLen, int callIntvl) {
        eventSet = new PriorityQueue<Event>();
        availableOperators = operators;
        avgCallLen = avgLen;
        freqOfCalls = callIntvl;
        nextCallTime = 0;
    }
    public void runSim(long stoppingTime) {
        //...
    }
}
```

CallSim Constructor

```java
/**
 * Constructor.
 * @param operator number of operators.
 * @param avgLen average length of a call.
 * @param callIntvl the average time between calls.
 */
public CallSim(int operators, double avgLen, int callIntvl) {
    eventSet = new PriorityQueue<Event>();
    availableOperators = operators;
    avgCallLen = avgLen;
    freqOfCalls = callIntvl;
    nextCallTime = 0;
    nextCall = new Event();
    // Schedule first call
}
```
```java
private int int_userNum = 0;
private int int_nextCallTime = 0;

/**
 * Place a new DIAL_IN event into the event queue.
 * Then advance the time when next DIAL_IN event will occur.
 * In practice, we would use a random number to set the time.
 */
private void void nextCall( int delta )
{
    Event ev = new Event( userNum++, nextCallTime, Event.DIAL_IN );
    eventSet.insert( ev );
    nextCallTime += delta;
}
```

```java
public void runSim( long stoppingTime )
{
    Event e = null;
    long howLong;
    while( !eventSet.isEmpty() )
    {
        e = eventSet.remove();
        if( e.time > stoppingTime )
            break;
        if( e.what == Event.HANG_UP ) // HANG_UP
        {
            availableOperators++;
            System.out.println( "User " + e.who + " hangs up at time " + e.time );
        }
        else // DIAL_IN
        {
            System.out.print( "User " + e.who + " dials in at time " + e.time + "" );
            if( availableOperators > 0 )
            {
                availableOperators--;
                howLong = nextPoisson( avgCallLen );
                System.out.println( "and connects for " + howLong + " minutes" );
                e.time += howLong;
                e.what = Event.HANG_UP;
                eventSet.add( e );
            }
            else
                System.out.println( "but gets busy signal!" );
        }
    }
}
```

```java
Random Number Generation Based on the Poisson Distribution
Not in JDK. Listed in Fig 9.5 on p. 404. We use r.nextDouble() for
nextDouble(), where r is a JDK Random.
```