Q1(a) 15 points:
Basic solution should identify the following aspects

(a) Reserve works as specified when it is successful
(b) Reserve does not work if attempt is made to overbook
(c) Failed reservations do not take up any seats

-11pts	if something relevant is written in english (the ceiling of deductions if something is written)
-9 pts	if the answer basically just explains the code without saying what it does
-7 pts	if the answer is vague, e.g. says "to check if reserve works correctly" (true but not specific enough) . Basically misses the nuance of a,b,c
-3 pts	if the answer says "to check if reserve allows small reservations and prevents overly large reservations and throws an exception" (misses c)
-0 pts	if the answer says "to check if reserve allows small reservations, prevents overly large reservations, and that a failed reservation doesn't take up any seats" (IF ANSWER IS PERFECT, MARK AS BONUS +0 SO THAT WE KNOW ITS GRADED)

(b) 10 points

-5pts	identifies reserveSeats as buggy but says nothing else
-5 pts	explains that partial reservations leave behind side-effects, but does not point to specific methods
-8 pts	Something relevant is written in english (ceiling of deductions if something is written)
-0 pts	Identifies reserveSeats as buggy, and explains that a reservation that is too big still reserves some seats (IF ANSWER IS PERFECT, MARK AS BONUS +0 SO THAT WE KNOW ITS GRADED)

(c) Basic solution: In BasicFlight::reserveSeats (the plural one), add a check before the loop:

```
if (availableSeats < names.size()) { throw new
IllegalStateException(...);}
```

-3 pts	explanation is vague and imprecise, but on correct lines
-7 pts	answer does not fix bug, or something relevant is written in english
-0 pts	explanation is clear and correct (or correct code is provided) (IF ANSWER IS PERFECT, MARK AS BONUS +0 SO THAT WE KNOW ITS GRADED)

Q2: 15 points

Solution: Test correctly sets up an Epsilon, adds Flights to it, tests that

(a) For a nonexistent flight, returns an empty list
(b) For multiple flights with the same start/end, returns all of them correctly
(c) For a single flight, returns a single flight correctly

-4 pts	nonexistent flight not tested
-5 pts	multiple same path flights not tested
-2 pts	problems with assert statement, proper test mechanics (AE for assertEquals is OK, using '' for identical code is OK)
-1 pts	@Test is missing
-2 pts	code is correct, but it is not a complete method (just body)
-12 pts	something relevant is written in english (the ceiling of all deductions if something is written)
-0 pts	All of the above cases are tested in complete JUnit method(s) (IF ANSWER IS PERFECT, MARK AS BONUS +0 SO THAT WE KNOW ITS GRADED)

Q3 (a) 10 points
Basic solution:

1. Add new interface to add method to get names on standby
2. Add new type of flight that extends BasicFlight (support existing flights) and implements above interface
3. Add field to store reservations on standby
4. Add new description for reservation status

-7 pts	something relevant written in english (the ceiling of all deductions if something is written)
-4 pts	Changes require modifying existing code, even if correct
-2 pts	all 4 items above present but answer is long winded and vague (essay type)
-1 pts	2, 3, 4 present. Method to return standby present but not in an interface
-2 pts	Exactly 1 missing from 2,3,4 above
-4 pts	exactly 2 items missing from 2, 3, 4
-0pts	All of the above changes are present, written in bullet/point form (IF ANSWER IS PERFECT, MARK AS BONUS +0 SO THAT WE KNOW ITS GRADED)

Q3 (b) 20 points Solution:

```
import java.util.ArrayList;
import java.util.List;

public class StandbyFlight extends BasicFlight {
  private final ArrayList<Reservation> standby;
  public StandbyFlight(int flightNo, String from, String to, int
capacity) {
      super(flightNo, from, to, capacity);
      this.standby = new ArrayList<>();
  }
  @Override
  public Reservation reserveSeats(List<String> names) {
     try {
          return super.reserveSeats(names);
     } catch(IllegalStateException e) {
          Reservation r = new Reservation(names,
this.getFlightNumber());
          this.standby.add(r);
          return r;
     }
  }

  @Override
  public void cancelReservation(Reservation r) {
     if (standby.contains(r))
          this.standby.remove(r);
     else {
          super.cancelReservation(r);
          for (int i = 0; i < standby.size(); i++) {
               Reservation s = standby.get(i);
               if (s.getPassengers().size() <=
this.getNumAvailableSeats()) {
                         for (String passenger : s.getPassengers())
                              this.reserveSeat(passenger);
                         this.standby.remove(i);
                         i--;
               }
          }
     }
  }

  @Override
  public String getConfirmationStatus(Reservation r) {
     if (this.standby.contains(r))
     return "On standby";
     return super.getConfirmationStatus(r);
  }

  public List<String> standbyPassengers() {
     ArrayList<String> ans = new ArrayList<>();
     for (Reservation r : this.standby)
     ans.addAll(r.getPassengers());
```

```
        return ans;
    }
}
```

Summary:

1. Need to have a ^List<Reservation> standby^
2. Need to override ^reserveSeats^, so that if it fails, it adds a ^Reservation^ to ^standby^
3. Need to override ^cancel^, to walk through ^standby^ an try to pull reservations off the list
4. Need to override ^getConfirmationStatus^ to add the new status
5. Need to implement ^getStandbyNames^

-2 pts	no list of standby reservations/ representation is not correct (any ordered collection should be OK)
-5 pts	Code is modified instead of extension (any code)
-4 pts	no code that adds standby reservations to the standby list
-2 pts	Code that adds standby reservations to the standby list is incorrect
-6 pts	no code that confirms reservations upon cancellation
-3 pts	Code that confirms reservations upon cancellation is incorrect (buggy, or does not check in order)
-2 pts	no code that gets confirmation status for a reservation
-16 pts	something relevant is written in english (the ceiling of all deductions if something is written). Use this deduction if answer is not in code form
-0 pts	all methods above implemented correctly OR some other way that makes all points in (b) work (IF ANSWER IS PERFECT, MARK AS BONUS +0 SO THAT WE KNOW ITS GRADED)

Q3(c) 5 points

Solution: The number of remaining seats on the ^Flight^ is smaller than the size of the smallest ^Reservation^ on the standby list.

-4 pts	something relevant is written in english (the ceiling of all deductions if something is written)

-2 pts	invariant captures the correct constraint but is not a logical statement about the state ("should be", etc.)
-0 pts	invariant is stated correctly (IF ANSWER IS PERFECT, MARK AS BONUS +0 SO THAT WE KNOW ITS GRADED)