1. I chose to implement best-fit because it seemed like it would be fun and I had a few basic ideas, though my final implementation feels far from perfect.

2. I created a list of instances of a custom Hole class that I sorted by size from smallest to largest.

3. The Hole class contains information regarding the address and size of each Hole that I used in a Map connecting PID and a submap of address and size. This allowed me to access each process by PID and get a collection of its segment addresses and sizes. I used the LinkedHashMap due to it retaining item order when indexed through[1], allowing me to ensure that text, data, and heap segments are listed properly without further complicating the data structure.

4. I was lazier with paging. I used a simple boolean array where true signifies that a page is being used and brute-force searched it for empty pages.

5. To track page mappings I used another nested class called Process which held a two dimensional int array. Index into the outer array represents virtual page number, the first index of the inner array holds the physical page number, and the latter index holds the number of bytes used in that page.

6. Paging has no external fragmentation due to the nature of the page structure. However, as a result, it accumulates far more internal fragmentation. On the other hand, pure segmentation has no internal fragmentation and, as such, my approximation of segmentation (which adds holes with 16 bytes or less to the segment currently being allocated) has much less internal fragmentation than paging. However, because segmented memory is not divided into equivalently sized segments, segmentation causes much greater external fragmentation.

7. The following test case has no internal fragmentation as every process either leaves greater than 16 bytes remaining in the holes it is placed in or completely fills said hole. This removes cases where the algorithm would include unused memory in a process. This test case also has no cases where a process cannot be allocated sufficient memory due to external fragmentation.
   1024 0
   A 234 1 80 100 54
   A 458 2 300 98 60
   A 300 3 150 80 70
   D 1
   P
   A 220 4 90 60 70

[1] http://stackoverflow.com/questions/683518/java-class-that-implements-map-and-keeps-insertion-order

A 890 5 200 450 240
D 3
A 170 6 105 5 60
D 2
P
A 90 7 40 40 10
D 5
D 4
A 345 8 50 110 185
A 128 9 64 32 32
A 128 10 64 32 32
D 7
D 6
A 245 11 120 64 61
P

8. The following test case is made entirely of processes whose byte counts are 1 over an even multiple of 32. As a result, the processes all only use 1 byte of their final page, leading to worst-case internal fragmentation
   1024 1
   A 161 1 80 100 54
   A 161 2 300 98 60
   A 161 3 150 80 70
   D 1
   P
   A 161 4 90 60 70
   A 161 5 200 450 240
   D 3
   A 161 6 105 5 60
   D 2
   P
   A 321 7 40 40 10
   D 5
   D 4
   A 321 8 50 110 185
   A 161 9 64 32 32
   A 225 10 64 32 32
   D 7
   D 6
   A 225 11 120 64 61
   P