1.

The topic of forcing declaration for primitives at time of initialization is largely based on how much we want to assist the programmer when using the language. If we trust the programmer to always initialize before using their variables, we can improve our readability and writability by avoiding forcing initial values that likely will never get used. Take the following Java input reading examples.

Example One:

```
int id = 0;
boolean graduate = true;
double currentGPA = 0.0;
Scanner sc = new Scanner(System.in);

System.out.println("Enter your id");
a = sc.nextInt();
System.out.println("Enter true if you are a graduate, false if you are not");
graduate = sc.nextBoolean();
System.out.println("Enter your current GPA");
currentGPA = sc.nextDouble();
```

Example Two:

```
int id; boolean graduate; double currentGPA;
Scanner sc = new Scanner(System.in);

System.out.println("Enter your id");
a = sc.nextInt();
System.out.println("Enter true if you are a graduate, false if you are not");
graduate = sc.nextBoolean();
System.out.println("Enter your current GPA");
currentGPA = sc.nextDouble();
```

Example One’s 337 characters against Example Two’s 320, yields an approximate 5% reduction in writing required. I would additionally argue the latter example without requiring declaration appears cleaner and more readable, with no unused values stored. Over a much larger system a 5% reduction will be more significant. However, this comes at a cost. Suppose our developer is not as experienced as we would like, and they attempt to write the following.

```
int a, b;
  for(int i = 0; i < 10; i++){
    b *= a;
  }
```

One may look at this trivial example and declare it to be a goofy mistake with little consequences, but what occurs when we don’t have proper initialization checking in place? Most modern compilers have protection against this sort of thing, but, if our language does not have anything in its compiler, we could hit a whole slew of buffer overflow errors. Small coding errors could bring our system to a halt in the worst case, but in a much less apocalyptic outlook could cause our programmers to spend more than they’d like on what amounts to a small, legal syntax error.

I propose a 7 out of 10 in favor of requiring declaration with each instantiation. A 5% decrease in writing paired with additional readability are legitimate improvements, knocking 3 points off a perfect 10 in my head. The dangers of buffer or memory overflow, or extended troubleshooting with what could be a simple syntax error reminder takes precedent in my mind.

2.

Output without pThread errors of Sample.java:

```
dunnnm2
dunnnm2  pts/33      75.187.67.101    Thu Jan 30 16:23   still logged in
dunnnm2  pts/41      75.187.67.101    Thu Jan 30 15:24 - 16:22  (00:57)
dunnnm2  pts/12      10.8.0.214       Thu Jan 30 11:13 - 13:19  (02:06)
dunnnm2  pts/12      10.8.0.214       Thu Jan 30 11:00 - 11:08  (00:08)
zmudam   pts/11      10.32.1.163      Tue Jan 28 11:50 - 11:51  (00:00)
zmudam   pts/10      10.32.1.163      Tue Jan 28 11:05 - 13:20  (02:15)
zmudam   pts/3       10.32.1.163      Sun Jan 12 11:37 - 11:38  (00:00)
zmudam   pts/3        10.32.1.163      Sun Jan 12 11:17 - 11:19  (00:01)
zmudam   pts/1        10.32.1.163      Sun Jan  5 09:58 - 10:29  (00:30)
```

3.

Insofar as I could discover, my code runs without any bugs I could find in my personal testing.

I opted to run a formalized run of 10 runs for 3 seperate programs, identical in structure in both Z+- in java, and averaged the results. The first test focuses on Integer manipulation. The second test focuses on boolean manipulation. The third test focuses on string manipulation.

Here are the results:

Test Type	Avg Z+- Runtime (seconds) over 10 runs	Avg Java Runtime (seconds) over 10 runs	Factor Java is faster by
Integer Manipulation	9.121	0.002	4560.5
Bool Manipulation	6.032	0.0007	8617.14
String Manipulation	33.766	26.005	1.298

The basic integer and boolean manipulation completely blew Integer and Bool manipulation out of the water. The string manipulations were done on a test that would have a reasonable runtime on Z+-, and went faster than I had anticipated. Given an even lengthier string run, I imagine the string manipulation tasks would have taken even longer. Java as a language uses immutable strings, so string manipulation in base java is an intensive task as is, but it very clearly has much more advanced optimization in place for Integer and Bool manipulation.