CS100J Prelim 3, Spring 2005 **Name: ____________** **Net ID: _____**

Look over the entire exam before starting. Don't spend too much time on one problem. If you get stuck on one problem, work on the others for a while and return to the difficult problem later, if you have time.

There are five pages.

Use the backs of pages if you need more room. You can unstaple the pages; we have a stapler at the front of the room.

0. (2 pts.) Write your name and net id —legibly— on every page of this exam.

0.		out of 02
1.		out of 20
2.		out of 12
3.		out of 20
4.		out of 20
5.		out of 26
6.		out of 100

1. (20 pts.) Write the body of the following method, as discussed in class and in the text and in the ProgramLive CD. Remember: we do not want to see an inner loop.

```
/** Sort array b in ascending order (smallest to largest), using selection sort */
public void selectionSort(int[] b) {









}
```

2. (12 pts.) Class `Inmate`: Study class `Inmate` and answer the questions given below.

```
/** An instance is a prison inmate, with a tattoo for an id */
public class Inmate {
   private int tattoo;  // the id of the inmate

   /** Constructor: an inmate with id id */
   public Inmate(int id) {
      tattoo= id;
   }

   /** = this inmate's id */
   public int getID() {
      return tattoo;
   }

   /** = "a and b are not null and have the same id" */
   public static boolean compare(Inmate a, Inmate b) {
      return (a != null && b != null && a.getID() == b.getID());
   }
}
```

a. Write a call of method `compare` to compare two `Inmates` `x` and `y`. The call does *not* occur within class `Inmate`.

b. Write a statement to create and insert a new `Inmate` with id 42 into a preexisting `Vector v`.

c. Write an assignment to store the first element of a `Vector v` into a newly declared variable `fred`. The assignment must include the declaration of `fred`, and it should not use a cast. What is the apparent type of variable `fred`?

d. Suppose the real type of variable `fred` (see part c) is `Inmate`. Write a statement that assigns `fred`'s id into a new **`int`** variable `x`.

3. (20 pts.) Circus-land is inhabited by clowns (see class `Clown` below) —almost too many to keep track. Fortunately, all of them used to be inmates in the New Jersey state penitentiary and thus have id numbers tattooed on their shoulders. A side effect of being in the Pen is that whenever they hear a whistle they scramble into a line for roll-call. See question 2 for class `Inmate.`

Your task is to complete static method `isSorted` (below), determining whether an array of `Clown`s are lined up in increasing order (by id). Writing a loop invariant for your loop will make our task of grading easier, but it is not required.

```
/** An instance is a clown */
public abstract class Clown extends Inmate {
   /** Constructor: an instance is a clown with identification id*/
   public Clown(int id) {
      super(id);
   }

   /** = something about this kind of Clown. E.g. if this clown has
     has three legs,the result might be "I have three legs!"*/
   public abstract String theClownyThing();

   /** = a new clown of the same kind as this one, with same id */
   public abstract Clown duplicateSelf();

   /** = a string representation of this clown, giving its kind and id */
   public String toString() {
      return  theClownyThing() + " and id " + getID();
   }

   /** = "lineup is in ascending order (by id)" */
   public static boolean isSorted(Clown[] lineup) {














   }

}
```

4. (20 pts.) There's no such thing as a normal `Clown`. That is why class `Clown` (of question 3) is an abstract class. For example, circus-land has red-nosed clowns and large-eared clowns. Write ONE non-abstract class, for red-nosed clowns, that extends class `Clown` of question 3 so that we can instantiate some `Clowns` from circus-land and be amused by their antics. Put in the minimum number of methods (constructors, procedures, functions) needed for the class definition to be legal Java.

5. (26 pts.) When groups of clowns get on a train, it is hard to tell which clowns have paid for their tickets because clowns keep duplicating themselves (see method `duplicateSelf` in question 3). Something must be done to eliminate the duplicates.

Write the body of methods `ejectClown` and `purgeTrain`, which deal with a `Vector train`. Each needs a loop. We give the postconditions of the method bodies and the loop invariants; your method body must use them. In writing the method bodies, consider these points:

• `train.size()` is the number of objects in `Vector train`.
• `train.remove(k)` removes element `k` from `Vector train`.
• All elements of `train` are of class `Clown` (or its subclasses).
• Use method `Clown.compare` (see question 2) to test whether two clowns have the same id.

(5a) `/**` Remove all `Clowns` from `train[i..]` that have the same id as `Clown target`.

```
                 Precondition: 0 ≤ i ≤ train.size() */
public static void ejectClown(Vector train, Clown target, int i) {
   // initialization:

   // invariant: no clown in train[i..k-1] has the same id as target

   // post: no clown in train[i..train.size()-1]  the same id as target
}
```

(5b) `/**` Remove duplicate clowns from train (so that no two clowns have the same id). `*/`

```
public static void purgeTrain(Vector train) {
   // initialization:

   // invariant: duplicates of train[0..h-1] have been removed from train

     // post: duplicates of train[0..train.size()-1] have been removed from train
}
```