```java
package ancestor;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

public class Main {

	public static int GENERATIONS = 66;
	public static int POPULATION = 100000;
	public static double avg;
	public static int desc;
	public static int nodesc;

	static List<Generation> era = new ArrayList<Generation>();
	static List<Integer> generationZero = new ArrayList<Integer>();

	public static void main(String[] args) {
		System.out.print("Populating...");
		populate();
		System.out.print(" done.\n");

		System.out.print("Tracing children...");
		trace();
		System.out.print(" done.\n");

		System.out.print("Calculating results...");
		calculate();
		System.out.print(" done.\n\n");

		System.out.println("Average of descendants: " + avg);
		System.out.println("Ancestors with most descendant: " + desc);
		System.out.println("Number of individuals with no descendants: " + nodesc);

		FileWriter fstream = null;
		try {fstream = new FileWriter("out.txt");
		} catch (IOException e1) {e1.printStackTrace();}

		BufferedWriter out = new BufferedWriter(fstream);
		try {
			for (int i = 0; i < generationZero.size(); i++) {
				out.write(String.valueOf(generationZero.get(i))+"\n");
			}
		} catch (IOException e) {e.printStackTrace();}
		try {out.close();} catch (IOException e) {e.printStackTrace();}

	}

	private static void trace() {
		for (int i = 0; i < Main.POPULATION; i++) {
			generationZero.add(0);
		}

		//Walk through existing population (id=65)
		for (int i = 0; i < Main.POPULATION; i++) {
			System.out.println("Individuum "+i);
			int gen = 0;
			Generation nextGeneration = era.get(gen);

			Set<Integer> current = new HashSet<Integer>();
			Set<Integer> children = new HashSet<Integer>();

			current.add(i);

			for (int j = 0; j < Main.GENERATIONS; j++) {
				nextGeneration = era.get(j);
```



```java
				//search for children
				for (int k = 0; k< Main.POPULATION; k++) {
					if (i==nextGeneration.getDad(k) || i==nextGeneration.getMom(k))
						children.add(k);
				}

				//next generation
				current.addAll(children);
				children.clear();
			}
			generationZero.set(i, current.size());
		}
	}

	private static void calculate() {
		for (int i = 0; i < generationZero.size(); i++) {
			int nrOfDescendants = generationZero.get(i);

			// Who has no descendants?
			if (nrOfDescendants==0) nodesc++;

			// Who has the most?
			if (nrOfDescendants > desc) desc = nrOfDescendants;

			avg+=nrOfDescendants;
		}
		avg /= generationZero.size();
	}

	private static void populate() {
		for (int i = 0; i < Main.GENERATIONS; i++) {
			era.add(new Generation(i));
		}
	}
}
```