Ages.java

```
import chn.util.*;
import apcslib.Format;

/**
 *  Sample program using an array of objects
 *
 * @author     G. Peck
 * @created    July 18, 2002
 */
public class Ages
{
  private Student[] studentList;

  /**
   *  Default Constructor for the Ages object. Invokes the
   *  Ages(String fName) constructor to load the file "names.txt"
   */
  public Ages()
  {
    this("names.txt");
  }

  /**
   *  Constructor for the Ages object. Loads Student records from file.
   *
   * @param  fName  File name containing student data
   */
  public Ages(String fName)
  {
    loadFile(fName);
  }

  /**
   *  Displays the contents of the Student array
   */
  public void displayList()
  {
    for (int index = 0; index < studentList.length; index++)
    {
      String wholeName = studentList[index].getLastName() + ", " +
          studentList[index].getFirstName();
      System.out.println(Format.right(index + 1, 3) + "  " +
          Format.left(wholeName, 15) +
          Format.right(studentList[index].getAge(), 5));
    }
    System.out.println();
  }
```

```
/**
 *  Stores the Student array data to disk. The first line of the file
 *  contains the number of Student records contained in the file. Each
 *  Student record consists of 2 Strings (lastName, firstName) and an
 *  integer (age).
 *
 * @param  outFileName  Name of output file
 */
public void saveFile(String outFileName)
{
  FileOutput outFile = new FileOutput(outFileName);

  outFile.println(studentList.length);
  for (int recNum = 0; recNum < studentList.length; recNum++)
  {
    outFile.println(studentList[recNum].getLastName() + " " +
        studentList[recNum].getFirstName() + " " +
        studentList[recNum].getAge());
  }
  outFile.close();
}

/**
 *  Helper method to invoke the recursive quicksort method
 */
public void Sort()
{
  quickSort(studentList, 0, studentList.length - 1);
}

/**
 *  Description of the Method
 *
 * @return    Description of the Returned Value
 */
public String toString()
{
  String report = "";

  for (int numStudents = 0; numStudents < studentList.length; numStudents++)
  {
    report += (numStudents + 1) + " " + studentList[numStudents].toString() + "\n";
  }

  return report;
}
```



```
/**
 *  Loads a file containing Student data. The first line of the file
 *  contains the number of Student records contained in the file.
 *  Each Student record consists of 2 Strings (lastName, firstName)
 *  and an integer (age).
 *
 * @param  inFileName  Name of file to be opened and read
 */
private void loadFile(String inFileName)
{
  int age;
  String lastName;
  String firstName;

  FileInput inFile = new FileInput(inFileName);

  int numStudents = inFile.readInt();
  studentList = new Student[numStudents];

  for (int recNum = 0; recNum < numStudents; recNum++)
  {
    lastName = inFile.readToken();
    firstName = inFile.readToken();
    age = inFile.readInt();
    studentList[recNum] = new Student(lastName, firstName, age);
  }
}
/**
 *  Sorts Student objects in an array between first index and last
 *  index using a quicksort algorithm.
 *
 * @param  list   an array of type Student
 * @param  first  start location in array to be sort
 * @param  last   end location in the array to be sorted
 */
private void quickSort(Student[] list, int first, int last)
{
  int g = first;
  int h = last;
  int midIndex;
  Student dividingValue;

  midIndex = (first + last) / 2;
  dividingValue = list[midIndex];
  do
  {
    while (list[g].compareTo(dividingValue) < 0){ g++; }
    while (list[h].compareTo(dividingValue) > 0){ h--; }
    if (g <= h)
    {
      // swap g and h
      Student temp = list[g];
      list[g] = list[h];
      list[h] = temp;
      g++;
      h--;
    }
  } while (g < h);
  if (h > first)
  {
    quickSort(list, first, h);
  }
  if (g < last)
  {
    quickSort(list, g, last);
  }
}
```



```
  /**
   *  Instantiates an Ages object, displays the contents, sorts the
   *  Student records by age, redisplays the contents, and saves the
   *  resulting sorted data to disk.
   *
   * @param  args  The command line arguments (not used)
   */
  public static void main(String[] args)
  {
    Ages studentData = new Ages("names20.txt");

    System.out.println("Unsorted List");
    System.out.println();
    studentData.displayList();

    studentData.Sort();

    System.out.println("List Sorted by Age");
    System.out.println();
    studentData.displayList();

    studentData.saveFile("sortedNames.txt");
  }
}

/**
 *  The Student class represents the last name, first name, and age
 *  of a student. methods are provided for constructing, accessing and
 *  comparing Students.
 *
 * @author     G. Peck
 * @created    July 18, 2002
 */
public class Student implements Comparable
{
  private String myFirstName;
  private String myLastName;
  private int myAge;

  /**
   *  Constructor for the Student object
   *
   * @param  lastName   Last name of Student
   * @param  firstName  First name of Student
   * @param  age        Age of Student
   */
  public Student(String lastName, String firstName, int age)
  {
    myLastName = lastName;
    myFirstName = firstName;
    myAge = age;
  }

  /**
   *  Gets the lastName attribute of the Student object
   *
   * @return    The lastName value
   */
  public String getLastName()
  {
    return myLastName;
  }
```



```
  /**
   *  Gets the firstName attribute of the Student object
   *
   * @return    The firstName value
   */
  public String getFirstName()
  {
    return myFirstName;
  }

  /**
   *  Gets the age attribute of the Student object
   *
   * @return    The age value
   */
  public int getAge()
  {
    return myAge;
  }

  /**
   *  Compares this Student object to another Student object
   *
   * @param  other        The object to be compared
   * @return              the value 0 if the argument is a Student whose age
   *                      field is equal to this Student; a value less than 0
   *                      if the argument is a Student whose age is greater
   *                      than this Student; and a value greater than 0 if the
   *                      argument if a Student whose age is less than this
   *                      Student.
   */
  public int compareTo(Object other)
  {
    return myAge - ((Student) other).myAge;
  }

  /**
   *  Compares this Student to the specified object. The result is true
   *  if and only if the argument is not null and is a Student object that
   *  has the same age value as this object.
   *
   * @param  otherObject  An object to compare this Student against
   * @return              true if the Student ages are equal; false otherwise.
   */
  public boolean equals(Object otherObject)
  {
    return compareTo(otherObject) == 0;
    //Student other = (Student) otherObject;
    //return myAge == other.myAge;
  }

  /**
   *  The Student object is returned as a String. overrides toString
   *  in class Object
   *
   * @return     String representation of the Student instance variables
   */
  public String toString()
  {
    return ("Last Name: " + myLastName +
            ", First Name: " + myFirstName +
            ", Age:  " + myAge);
  }
}
```

