CIT 594 Data Structures & Software Design: Modularity

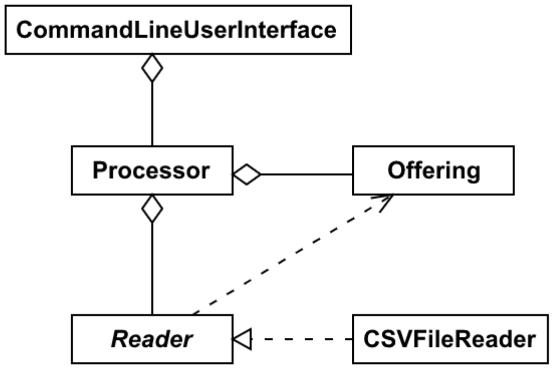
The following code is an example of a “Monolith” approach in which all functionality is contained in a single method:

```java
public class Monolith {

    public static void main(String[] args) {

        if (args.length < 1) {
            System.out.println("Usage: java Monolith [filename]");
            System.exit(0);
        }

        String filename = args[0];

        // read the file into a data structure
        ArrayList<ArrayList<String>> offerings = new
                                    ArrayList<ArrayList<String>>();
        Scanner in = null;
        try {
            in = new Scanner(new File(filename));
            in.useDelimiter(",");
            while (in.hasNext()) {
                ArrayList<String> offering = new ArrayList<String>();
                offering.add(in.next()); // course number
                offering.add(in.next()); // instructor
                offering.add(in.next()); // enrollment
                in.nextLine(); // consume the rest of the line
                offerings.add(offering);
            }
        }
        catch (Exception e) {
            System.out.println("Exception reading input file");
            e.printStackTrace();
        }
        finally {
            in.close();
        }

        // ask the user what they want to do
        System.out.print("Enter 1 to search by course, 2 to search by
                         instructor, 3 for average for a course: ");
        in = new Scanner(System.in);
        int choice = in.nextInt();
        if (choice == 1) {
            System.out.print("Enter the course number: ");
            String course = in.next();
            HashSet<String> instructors = new HashSet<String>();
            for (ArrayList<String> offering : offerings) {
                if (offering.get(0).contains(course)) {
                    instructors.add(offering.get(1));
                }
```

```java
                  }
                  System.out.println("Here are the instructors of the course:");
                  for (String instructor : instructors) {
                        System.out.println(instructor);
                  }
            }
            else if (choice == 2) {
                  System.out.print("Enter the instructor name: ");
                  String instructor = in.next();
                  HashSet<String> courses = new HashSet<String>();
                  for (ArrayList<String> offering : offerings) {
                        if
           (offering.get(1).toUpperCase().contains(instructor.toUpperCase())) {
                              courses.add(offering.get(0));
                        }
                  }
                  System.out.println("Here are the courses taught by that
                                                                instructor:");
                  for (String course : courses) {
                        System.out.println(course);
                  }
            }
            else if (choice == 3) {
                  System.out.print("Enter the course number: ");
                  String course = in.next();
                  int total = 0;
                  int count = 0;
                  for (ArrayList<String> offering : offerings) {
                        if (offering.get(0).contains(course)) {
                              count++;
                              total += Integer.parseInt(offering.get(2).trim());
                        }
                  }
                  if (count > 0) {
                        double average = ((double)total)/count;
                        System.out.println("Average enrollment: " + average);
                  }
                  else {
                        System.out.println("That course has not been taught");
                  }
            }
            in.close();
      }
}
```

The code can be made more **modular** by using an N-tier architecture and using different components/modules/classes to handle different functionality.

Here is a UML class diagram for the solution; the implementation of each class is shown on the following pages:



This first class shown here represents the code in the **User Interface** or Presentation tier. It is responsible for interacting with the user and modifying data only as it relates to displaying it.

It depends *only* on the class(es) in the Processor/Logic tier and *not* any class(es) in the Data Management tier.

Note that this code is in the **edu.upenn.cit594.ui** package to indicate that it is part of the User Interface tier. Putting this code into a separate package is not required by Java, but it helps the reader of the code understand its purpose.

```
package edu.upenn.cit594.ui;

public class CommandLineUserInterface {

      protected Processor processor;
      protected Scanner in;

      public CommandLineUserInterface(Processor processor) {
            this.processor = processor;
            in = new Scanner(System.in);
      }

      public void start() {
            System.out.print("Enter 1 to search by course, 2 to search by
                                    instructor, 3 for average for a course: ");
            int choice = in.nextInt();
            if (choice == 1) {
                  doInstructorsForCourse();
            }
            else if (choice == 2) {
                  doCoursesForInstructor();
            }
            else if (choice == 3) {
                  doAverageEnrollmentForCourse();
            }
            in.close();
      }

      protected void doInstructorsForCourse() {
            System.out.print("Enter the course number: ");
            String course = in.next();
            Set<String> instructors = processor.getInstructorsForCourse(course);
            System.out.println("Here are the instructors of the course:");
            for (String instructor : instructors) {
                  System.out.println(instructor);
            }
      }

      protected void doCoursesForInstructor() {
            System.out.print("Enter the instructor name: ");
            String instructor = in.next();
            Set<String> courses = processor.getCoursesForInstructor(instructor);
```

```java
		System.out.println("Here are courses taught by that instructor:");
		for (String course : courses) {
			System.out.println(course);
		}
	}

	protected void doAverageEnrollmentForCourse() {
		System.out.print("Enter the course number: ");
		String course = in.next();
		double average = processor.getAverageEnrollmentForCourse(course);
		if (average >= 0) {
			System.out.println("Average enrollment: " + average);
		}
		else {
			System.out.println("That course has not been taught");
		}
	}
}
```

This next class represents a single course offering in the program. It is placed in the **edu.upenn.cit594.data** package to indicate that it is not part of any one tier, but rather is data that is shared by classes in different tiers.

```java
package edu.upenn.cit594.data;

public class Offering {

	private final String courseNumber;
	private final String instructor;
	private final int enrollment;

	public Offering(String courseNumber, String instructor, int enrollment) {
		this.courseNumber = courseNumber;
		this.instructor = instructor;
		this.enrollment = enrollment;
	}

	public String getCourseNumber() { return courseNumber; }
	public String getInstructor() { return instructor; }
	public int getEnrollment() { return enrollment; }
}
```

This next class represents the code in the **Processor** or Logic tier. It is responsible for processing data, e.g. filtering, sorting, etc. However, it should not contain any code for interacting with the user, or for reading from or writing to a data source.

It depends *only* on the class(es) in the Data Management tier and *not* on any class(es) in the User Interface tier.

Note that this code is in the **edu.upenn.cit594.processor** package to indicate that it is part of the Processor tier.

```java
package edu.upenn.cit594.processor;

public class Processor {

	protected Reader reader;
	protected List<Offering> offerings;

	public Processor(Reader reader) {
		this.reader = reader;
		offerings = reader.getAllOfferings();
	}

	public Set<String> getInstructorsForCourse(String course) {
		Set<String> instructors = new HashSet<String>();
		for (Offering offering : offerings) {
			if (offering.getCourseNumber().contains(course)) {
				instructors.add(offering.getInstructor());
			}
		}
		return instructors;
	}

	public Set<String> getCoursesForInstructor(String instructor) {
		Set<String> courses = new HashSet<String>();
		for (Offering offering : offerings) {
			if (offering.getInstructor().toUpperCase().
					contains(instructor.toUpperCase())) {
				courses.add(offering.getCourseNumber());
			}
		}
		return courses;
	}

	public double getAverageEnrollmentForCourse(String course) {
		int total = 0;
		int count = 0;
		for (Offering offering : offerings) {
			if (offering.getCourseNumber().contains(course)) {
				count++;
				total += offering.getEnrollment();
			}
		}
```

```
			if (count > 0) {
				return ((double)total)/count;
			}
			else return -1;
		}
}
```

The code below is part of the **Data Management** tier, which is responsible for reading data from and writing data to the data source.

Note that classes in this tier do not have dependencies on any class(es) in the User Interface or Processor tier.

This code is in the **edu.upenn.cit594.datamanagement** package to indicate that it is part of the Data Management tier. Note that this is different from the "edu.upenn.cit594.data" tier, which holds classes that are shared between tiers.

In this particular case, "Reader" is an interface and the "Processor" class only has a dependency on this interface, not any specific implementation. This is not necessarily part of the N-Tier Architecture but provides some flexibility in the design.

```
package edu.upenn.cit594.datamanagement;

public interface Reader {

	public List<Offering> getAllOfferings();

}
```

The next class is also part of the Data Management tier and implements the "Reader" interface. It is also in the **edu.upenn.cit594.datamanagement** package.

```
package edu.upenn.cit594.datamanagement;

public class CSVFileReader implements Reader {

	protected String filename;

	public CSVFileReader(String name) {
		filename = name;
	}

	public List<Offering> getAllOfferings() {
		List<Offering> offerings = new ArrayList<Offering>();
		Scanner in = null;
		try {
			in = new Scanner(new File(filename));
```

```java
                in.useDelimiter(",");
                while (in.hasNext()) {
                    String courseNumber = in.next();
                    String instructor = in.next();
                    int enrollment = Integer.parseInt(in.next().trim());
                    in.nextLine(); // consume the rest of the line
                    offerings.add(new Offering(courseNumber, instructor,
                                                            enrollment));
                }
            }
            catch (Exception e) {
                throw new IllegalStateException(e);
            }
            finally {
                in.close();
            }
            return offerings;
        }
}
```

The last class we see is the "Main" class. It is *not* part of any tier in the N-Tier Architecture, but rather its purpose is to create the objects in the other tiers and then set up their relationships, e.g. by passing one object to the constructor of another, and then start the application via the User Interface.

```java
package edu.upenn.cit594;

public class Main {

        public static void main(String[] args) {

            if (args.length < 1) {
                System.out.println("Usage: java Main [filename]");
                System.exit(0);
            }
            String filename = args[0];

            Reader reader = new CSVFileReader(filename);

            Processor processor = new Processor(reader);

            CommandLineUserInterface ui = new
                                    CommandLineUserInterface(processor);

            ui.start();

        }
}
```