Lab 17 – Comparator & Comparable

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class Lab16Main {
	public ArrayList<Movie> movieList = new ArrayList<Movie>();
	private static Scanner keyboard = new Scanner(System.in);
   public static final int MOVIE_COUNT = 5; // Return this many movies in the searches

	private Map<String, Movie> byNameMap = new HashMap<>();
	private Map<Integer, ArrayList<Movie>> byYearMap = new HashMap<>();
	private Map<String, ArrayList<Movie>> byGenreMap = new HashMap<>();

   /*
     Partially complete
     Add your code where needed
   */

	public static void main(String[] args) {
		Lab16Main lab16main = new Lab16Main();
		lab16main.readMovies("movies.tsv");
		int choice;
		do {
			choice = getMenuChoice();
			switch (choice) {
      // case 1 is done for you
				case 1: lab16main.sortBy("ID");
					lab16main.displayMovies(lab16main.getList());
					break;
				case 2:
			lab16main.sortBy("Name");
			lab16main.displayMovies(lab16main.getList());
					break;
				case 3:
			// Your code here
					lab16main.sortBy("Year");
					lab16main.displayMovies(lab16main.getList());
					break;
				case 4:
			// Your code here
					lab16main.sortBy("ReverseYear");
					lab16main.displayMovies(lab16main.getList());
					break;
				case 5:
					System.out.print("Enter the movie name: ");
			// Use nextLine() everywhere!
					String name = keyboard.nextLine();
			// Do something with name

	lab16main.displayMovies(lab16main.searchByName(name));
					break;
				case 6:
			// Your code here
					System.out.print("Enter the movie year: ");
					int year = Integer.parseInt(keyboard.nextLine());
					lab16main.displayMovies(lab16main.searchByYear(year));
					break;
				case 7:
			// Your code here
```

```
                                        System.out.print("Enter the movie genre: ");
                                        String genre = keyboard.nextLine();

        lab16main.displayMovies(lab16main.searchByGenre(genre));
                                        break;
                                case 8:
                                        lab16main.displayTotals();
                                        System.out.println("Bye");
                                        break;
                        }
                } while (choice != 8);
        }

   /*
     Don't change this method
   */
        public static int getMenuChoice() {
                System.out.println("1. Display by ID\n2. Display by name\n3. Display by year\n"
+
                                "4. Display by year in reverse\n5. Search by name\n6. Search by
year\n" +
                                "7. Search by genre\n8. Quit");
                System.out.print("Enter your choice: ");
                int choice = Integer.parseInt(keyboard.nextLine());
                if (choice < 1 || choice > 8) choice = 8;
                return choice;
        }

        public void readMovies(String filename) { // Lab 17
                Scanner fileInput = null;

                try {
                        fileInput = new Scanner(new File(filename));
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }

                if (fileInput != null) {
                        while (fileInput.hasNextLine()) {
                                String line = fileInput.nextLine();
                                String[] fields = line.split("\t");
                                Movie movie = toMovie(fields);
                                movieList.add(movie);

                                byNameMap.put(movie.getMovieName(), movie);

                                byYearMap.computeIfAbsent(movie.getYear(), k -> new
ArrayList<>()).add(movie);

                                for (String genre : movie.getGenres()) {
                                        byGenreMap.computeIfAbsent(genre, k -> new
ArrayList<>()).add(movie);
                                }
                        }
                        fileInput.close();
                }
        }

//      public void readMovies(String filename) { // Lab 16
//              Scanner fileInput = null;
//
```

```
//		try {
//			fileInput = new Scanner(new File(filename));
//		} catch (FileNotFoundException e) {
//			e.printStackTrace();
//		}
//
//	/* Your code here
//	   While there are lines in the file
//	    read a line and split it on \t
//	    create a Movie using toMovie
//	    add it to movieList
//	 */
//
//		if (fileInput != null)	{
//			while(fileInput.hasNextLine())		{
//				String line = fileInput.nextLine();
//				String[] fields = line.split("\t");
//				Movie movie = toMovie(fields);
//				movieList.add(movie);
//			}
//		}
//	 }

	public Movie toMovie(String[] str) {
	/*  Returns one Movie from the data in str
	    Each line of str should contain one field
	    Change last one to ArrayList<String> for genres
	*/
		int movieID = Integer.parseInt(str[0].strip());
		String movieName = str[1].strip().replace("\"", "");
		int year = Integer.parseInt(str[2].strip());
		String country = str[3].strip().replace("\"", "");
		ArrayList<String> genres = new ArrayList<String>();
	// Start at position #4
	// Strip and add to the ArrayList of genres
		for (int i=4; i<str.length; i++) {
			genres.add(str[i].strip());
		}
		return new Movie(movieID, movieName, year, country, genres);
	}

   // Don't change this, even if you don't like my table spacing
	private void displayMovies(ArrayList<Movie> list) {
		if (list.size() == 0) {
			System.out.println("Nothing to display");
		} else {
			System.out.format("%7s %50s %5s %30s %6s\n", "ID", "Name", "Year",
"Country", "Genres");
			for (Movie m: list) {
				System.out.format("%7s %50s %5d %30s ", m.getMovieID(),
m.getMovieName(),
					m.getYear(), m.getCountry());
				for (int i=0; i<m.getGenres().size(); i++) {
					System.out.print(m.getGenres().get(i) + " ");
				}
				System.out.println();
			}
		}
		System.out.println();
	}

   // Sort according to the field indicated by s
```

```
    public void sortBy(String s) {
        switch (s) {
            case "ID":
                // Use Movie's built-in compareTo
                Collections.sort(movieList);
                break;
            case "Name":
                // Your code here
                                Collections.sort(movieList,
Comparator.comparing(Movie::getMovieName));
                break;
            case "Year":
                // Your code
                                Collections.sort(movieList,
Comparator.comparing(Movie::getYear));
                break;
            case "ReverseYear":
                // Your code here
                                Comparator<Movie> reverseYears = Collections.reverseOrder(new
SortByYear());
                                Collections.sort(movieList, reverseYears);
                break;
        }
    }

    public ArrayList<Movie> searchByName(String name) { // Lab 17
            ArrayList<Movie> list = new ArrayList<>();
            Movie movie = byNameMap.get(name);
            if (movie != null) list.add(movie);
            return list;
    }

// Search for MOVIE_COUNT movies by name
//     public ArrayList<Movie> searchByName(String name) { // Lab 16
//     // Sort by id before searches for consistent results
//     sortBy("ID");
//
//     // List of results
//             ArrayList<Movie> list = new ArrayList<Movie>();
//
//     // Count the # of matches
//             int count = 0;
//             for (Movie m: movieList) {
//       // Does m match on the name key?
//                     if (m.getMovieName().equals(name)) {
//           // Yes, so add it to the result list
//                             list.add(m);
//                             count++;
//
//           // Quit if we hit the maximum # of movies to return
//                             if (count == Lab16Main.MOVIE_COUNT) break;
//                     }
//             }
//             return list;
//     }

    public ArrayList<Movie> searchByYear(int year) {        // Lab 17
            return byYearMap.getOrDefault(year, new ArrayList<>());
    }

// Search for MOVIE_COUNT movies by year
```

```
//	public ArrayList<Movie> searchByYear(int year) {	// Lab 16
//		ArrayList<Movie> list = new ArrayList<Movie>();
//
//	// Fill up list with MOVIE_COUNT movies that match on year
//	// Your code here
//		int count = 0;
//		for (Movie m: movieList) {
//			if (m.getYear() == year) {
//				list.add(m);
//				count++;
//				if (count == Lab16Main.MOVIE_COUNT) break;
//			}
//		}
//
//		return list;
//	}

	public ArrayList<Movie> searchByGenre(String genre) { // Lab 17
		return byGenreMap.getOrDefault(genre, new ArrayList<>());
	}

	// Search for MOVIE_COUNT movies by genre
//	public ArrayList<Movie> searchByGenre(String genre) {// Lab 16
//		ArrayList<Movie> list = new ArrayList<Movie>();
//
//	// Fill up list with MOVIE_COUNT movies that match on genre
//	// Your code here
//		int count = 0;
//		for (Movie m: movieList) {
//			if (m.getGenres().contains(genre)) {
//				list.add(m);
//				count++;
//				if (count == Lab16Main.MOVIE_COUNT) break;
//			}
//		}
//
//		return list;
//	}

	public void displayTotals() {
		System.out.println("Movie totals");
		System.out.println("movieList size : " + movieList.size());
		System.out.println("byNameMap size : " + byNameMap.size());
		System.out.println("byYearMap size : " + byYearMap.size());
		System.out.println("byGenreMap size : " + byGenreMap.size());
	}

	// Breaks encapsulation, boo!
	public ArrayList<Movie> getList() { return movieList; }

}
```

```java
import java.util.ArrayList;

public class Movie implements Comparable<Movie> {
	private int movieID;
	private String movieName;
	private int year;
	private String country;
	private ArrayList<String> genres;

   // Overloaded constructor
	public Movie(int movieID, String movieName, int year, String country, ArrayList<String>
genres ) {
		this.movieID = movieID;
		this.movieName = movieName;
		this.year = year;
		this.country = country;
		this.genres = genres;
	}

   // Don't change this
	public String toString() {
		StringBuilder sb = new StringBuilder();
		for (String g: genres) {
			sb.append(g + " ");
		}
		return movieID + " " + movieName + " " + year +
				" " + country + " " + sb.toString().strip();
	}

   public int getMovieID() { return movieID; }
	public String getMovieName() { return movieName; }
	public int getYear() { return year; }
	public String getCountry() { return country; }
	public ArrayList<String> getGenres() { return genres; }


   // Compare on movieID
	public int compareTo(Movie two) {
      // Your code here
      return Integer.compare(this.movieID, two.movieID);
	}

}

// Name: Shlok Kalekar
// Andrew ID: skalekar

import java.util.Comparator;

public class SortByYear implements Comparator<Movie> {

   @Override
   public int compare(Movie movie1, Movie movie2) {
      return Integer.compare(movie1.getYear(), movie2.getYear());
   }
}
```

Lab 18 – Queue & Enum

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.*;
import java.util.regex.Pattern;

public class Lab18 {
   public static ArrayList<Customer> readCustomers(String filename)
         throws IOException {

      // Array list of Customer objects
      ArrayList<Customer> list = new ArrayList<>();
      // Create a new File object
      File file = new File(filename);
      // If the file does not exist, throw a FileNotFound exception
      if (!file.exists())
         throw new FileNotFoundException(filename + " not found");
      // Create a new Scanner on the file object
      Scanner fileScanner  = new Scanner(file);
      // While fileScanner has a next line
      while(fileScanner.hasNextLine())    {
         int rating;
         double balance;
         // Read the next line and split it
         String line = fileScanner.nextLine();
         String[] fields = line.split(",");

         String firstName = fields[0];
         String lastName = fields[1];

         // Convert the rating to an int; throw NumberFormatException if bad
         try {
            rating = Integer.parseInt(fields[2].trim());
         } catch (NumberFormatException e)   {
            System.out.println("Invalid format for rating, an integer should be used!");
            rating = 0;
         }
         // Convert the balance to a double; throw NumberFormatException if bad
         try {
            balance = Double.parseDouble(fields[3].trim());
         } catch (NumberFormatException e)   {
            System.out.println("Invalid format for balance, a double should be used!");
            balance = 0.0d;
         }
         // Create a new customer object, add it to list
         Customer customer = new Customer(firstName, lastName, rating, balance);
         list.add(customer);
      }
      return list;
   }

   public static void main(String[] args) {
      ArrayList<Customer> clist = null;
      // Problem 3
      // Call readCustomers with the data file as a parameter
      try {
         clist = readCustomers("customers.csv");
      } catch (IOException e) {
```

```
            System.out.println(e.getMessage());
            return;
        }

        // Print the array list
        System.out.println("Original list");
        for (Customer customer : clist) {
            System.out.println(customer);
        }

        // Problem #4
        // Create PriorityQueue queue1
        PriorityQueue<Customer> queue1 = new PriorityQueue<>();
        try {
            for (Customer customer : clist) {
                queue1.add(customer);
            }
        } catch (Exception e) {
            System.out.println("Error while adding to queue1: " + e.getMessage());
        }

        // Problem #5
        // Create PriorityQueue queue2
        PriorityQueue<Customer> queue2 = new
PriorityQueue<>(Comparator.comparingDouble(Customer::getBalance));
        try {
            for (Customer customer : clist) {
                queue2.add(customer);
            }
        } catch (Exception e) {
            System.out.println("Error while adding to queue2: " + e.getMessage());
        }

        // Problem #6
        // Remove things one at a time from queue1 and print them
        System.out.println("Queue1 processing");
        while (true) {
            try {
                Customer customer = queue1.element();
                System.out.println(customer);
                queue1.remove();
            } catch (Exception e) {
                System.out.println("Done");
                break;
            }
        }

        // Problem #7
        // Remove things one at a time from queue2 and print them
        System.out.println("Queue2 processing");
        while (true) {
            try {
                Customer customer = queue2.element();
                System.out.println(customer);
                queue2.remove();
            } catch (Exception e) {
                System.out.println("Done");
                break;
            }
        }
```

```
        // Problem #8
        // Try this on your own

        // Problem #9
        problem9();
    }

    public static void problem9() {
        ArrayList<String> lines = new ArrayList<>();
        ArrayList<String> patterns = new ArrayList<>();

        try (Scanner scanner = new Scanner(new File("testdata.txt"))) {
            while (scanner.hasNextLine()) {
                lines.add(scanner.nextLine());
            }
        } catch (FileNotFoundException e) {
            System.out.println("testdata.txt not found.");
            return;
        }

        System.out.println("Contents of testdata.txt:");
        for (String line : lines) {
            System.out.println(line);
        }

        // Given patterns-
        patterns.add("\\d"); // any digit
        patterns.add("[a-zA-Z]"); // any letter, either case
        patterns.add("\\b\\d+\\b"); // an integer
        patterns.add("^a"); // starts with "a"
        patterns.add("s$"); // ends with "s"
        patterns.add("\\("); // contains a left parenthesis
        patterns.add("a.*e|e.*a"); // contains "a" and "e" in either order
        patterns.add("aeiou"); // contains vowels a, e, i, o, u in sequence
        patterns.add("a.*e.*i.*o.*u"); // contains vowels a, e, i, o, u in order, not necessarily
together

        for (String patternString : patterns) {
            System.out.println("\nPattern: " + patternString);
            Pattern pattern = Pattern.compile(patternString);

            for (String line : lines) {
                if (pattern.matcher(line).find()) {
                    System.out.println("Matched: " + line);
                }
            }
        }
    }
}

public enum RatingType { //RatingType rating = LOW;
    LOW(“THIS IS NOT HERE REALLY”), // this.rating = RatingType LOW;
    MEDIUM(“FOR THE TEST”),
    HIGH(“COOL”);
        private String description;
        private RatingType(String description) {this.description=description;}
        public String getDescription() {return description;} //rating.getDescription()
}
```

Lab 19 – I/O Functions

```
import java.io.*;
import java.util.Scanner;

public class Lab19Main {

    public static long printWriterTest(String filename, int n)  {
        long startTime = System.nanoTime();

        try (PrintWriter writer = new PrintWriter(
                new BufferedWriter(
                        new FileWriter(filename)))) {
            for (int i = 1; i <= n; i++) {
                writer.print('A');
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }

        return System.nanoTime() - startTime;
    }

    public static long bufferWriterTest(String filename, int n)  {
        long startTime = System.nanoTime();

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
            for (int i = 1; i <= n; i++) {
                writer.write('A');
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }

        return System.nanoTime() - startTime;
    }

    public static long fileWriterTest(String filename, int n)  {
        long startTime = System.nanoTime();

        try (FileWriter writer = new FileWriter(filename)) {
            for (int i = 1; i <= n; i++) {
                writer.write('A'); // Write 'A' using FileWriter
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }

        return System.nanoTime() - startTime;
    }

    public static long scannerTest(String filename, int n)  {
        long startTime = System.nanoTime();

        try (Scanner scanner = new Scanner(
                new BufferedReader(
                        new FileReader(filename)))) {
            scanner.useDelimiter("");
            for (int i = 1; i <= n && scanner.hasNext(); i++) {
```

```
            char ch = scanner.next().charAt(0);
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }

    return System.nanoTime() - startTime;
}

public static long bufferedReaderTest(String filename, int n)  {
    long startTime = System.nanoTime();

    try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
        for (int i = 1; i <= n; i++) {
            reader.read();
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }

    return System.nanoTime() - startTime;
}

public static long fileReaderTest(String filename, int n)  {
    long startTime = System.nanoTime();

    try (FileReader reader = new FileReader(filename)) {
        for (int i = 1; i <= n; i++) {
            reader.read();
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }

    return System.nanoTime() - startTime;
}

public static void main(String[] args) {
    String filename = "test.txt";

    // First test with 10000
    int n1 = 10000;
    System.out.printf("Testing with n = %d\n", n1);

    long printWriterTime1 = printWriterTest(filename, n1);
    long bufferWriterTime1 = bufferWriterTest(filename, n1);
    long fileWriterTime1 = fileWriterTest(filename, n1);
    long scannerTime1 = scannerTest(filename, n1);
    long bufferedReaderTime1 = bufferedReaderTest(filename, n1);
    long fileReaderTime1 = fileReaderTest(filename, n1);

    System.out.printf("%-20s %15d ns\n", "PrintWriter Test:", printWriterTime1);
    System.out.printf("%-20s %15d ns\n", "BufferedWriter Test:", bufferWriterTime1);
    System.out.printf("%-20s %15d ns\n", "FileWriter Test:", fileWriterTime1);
    System.out.printf("%-20s %15d ns\n", "Scanner Test:", scannerTime1);
    System.out.printf("%-20s %15d ns\n", "BufferedReader Test:", bufferedReaderTime1);
    System.out.printf("%-20s %15d ns\n", "FileReader Test:", fileReaderTime1);

    // Second test with 1000000
    int n2 = 1000000;
```

```
        System.out.printf("\nTesting with n = %d\n", n2);

        long printWriterTime2 = printWriterTest(filename, n2);
        long bufferWriterTime2 = bufferWriterTest(filename, n2);
        long fileWriterTime2 = fileWriterTest(filename, n2);
        long scannerTime2 = scannerTest(filename, n2);
        long bufferedReaderTime2 = bufferedReaderTest(filename, n2);
        long fileReaderTime2 = fileReaderTest(filename, n2);

        System.out.printf("%-20s %15d ns\n", "PrintWriter Test:", printWriterTime2);
        System.out.printf("%-20s %15d ns\n", "BufferedWriter Test:", bufferWriterTime2);
        System.out.printf("%-20s %15d ns\n", "FileWriter Test:", fileWriterTime2);
        System.out.printf("%-20s %15d ns\n", "Scanner Test:", scannerTime2);
        System.out.printf("%-20s %15d ns\n", "BufferedReader Test:", bufferedReaderTime2);
        System.out.printf("%-20s %15d ns\n", "FileReader Test:", fileReaderTime2);
    }
}
```

```
Testing with n = 10000
PrintWriter Test:            2957439 ns
BufferedWriter Test:         1520089 ns
FileWriter Test:            15184434 ns
Scanner Test:               35341861 ns
BufferedReader Test:         1643678 ns
FileReader Test:            10026304 ns

Testing with n = 1000000
PrintWriter Test:           40861088 ns
BufferedWriter Test:        26412169 ns
FileWriter Test:           101649826 ns
Scanner Test:              465852713 ns
BufferedReader Test:        25177802 ns
FileReader Test:            56890587 ns
```

Lab 20 – File Operations

```
import java.io.*;
import java.nio.file.*;
import java.util.ArrayList;
public class CargoFileOperations {
    private String filename;

    public CargoFileOperations(String filename) {
        this.filename = filename;
    }
    public void writeList(ArrayList<Cargo> list) {
        try (FileOutputStream fileOut = new FileOutputStream(filename);
             ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            for (Cargo cargo : list) {
                try {
                    out.writeObject(cargo);
                } catch (IOException e) {
                    System.err.println("Failed to write cargo: " + e.getMessage());
                }
            }
        } catch (IOException e) {
            System.err.println("Output file failed to open: " + e.getMessage());
            System.exit(1);
        }
    }
    public ArrayList<Cargo> readList() {
        ArrayList<Cargo> cargoList = new ArrayList<>();
        try (FileInputStream fileIn = new FileInputStream(filename);
             ObjectInputStream in = new ObjectInputStream(fileIn)) {
            while (fileIn.available() > 0) {
                try {
                    Cargo cargo = (Cargo) in.readObject();
                    cargoList.add(cargo);
                } catch (ClassNotFoundException | IOException e) {
                    System.err.println("Failed to read cargo: " + e.getMessage());
                }
            }
        } catch (IOException e) {
            System.err.println("Input file failed to open: " + e.getMessage());
            System.exit(1);
        }
        return cargoList;
    }
    // Method to display file information
    public void display() {
        Path path = Paths.get(filename);
        File file = path.toFile();
        System.out.println("Path: " + path.toString());
        System.out.println("Absolute Path: " + path.toAbsolutePath());
        System.out.println("Root: " + path.getRoot());

        System.out.println("File is directory: " + file.isDirectory());
        System.out.println("File absolute path: " + file.getAbsolutePath());

        System.out.println("isExecutable returns " + file.canExecute());
        System.out.println("isReadable returns " + file.canRead());
        System.out.println("isWritable returns" + file.canWrite());
    }
}
```

Lab 21 – Reflections

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Lab21Main {

    public void classFun(Class<?> c) {
        try {
            System.out.println("1. Canonical Class Name: " + c.getCanonicalName());

            System.out.println("\n2. Member Fields:\n");
            Field[] fields = c.getDeclaredFields();
            if (fields.length == 0) {
                System.out.println("No member data found!");
            } else {
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < fields.length; i++) {
                    sb.append("\t").append((i + 1)).append(". ").append(fields[i]).append("\n");
                }
                System.out.println(sb.toString());
            }

            System.out.println("\n3. Local Constructors: ");
            Constructor<?>[] localConstructors = c.getDeclaredConstructors();
            if (localConstructors.length == 0) {
                System.out.println("No local constructors found!");
            } else {
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < localConstructors.length; i++) {
                    sb.append("\t").append((i + 1)).append(".
").append(localConstructors[i]).append("\n");
                }
                System.out.println(sb.toString());
            }

            System.out.println("\n4. Public Constructors: ");
            Constructor<?>[] publicConstructors = c.getConstructors();
            if (publicConstructors.length == 0) {
                System.out.println("No public constructors found!");
            } else {
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < publicConstructors.length; i++) {
                    sb.append("\t").append((i + 1)).append(".
").append(publicConstructors[i]).append("\n");
                }
                System.out.println(sb.toString());
            }

            System.out.println("\n5. Local Methods: ");
            Method[] localMethods = c.getDeclaredMethods();
            if (localMethods.length == 0) {
                System.out.println("No local methods found!");
            } else {
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < localMethods.length; i++) {
                    sb.append("\t").append((i + 1)).append(". ").append(localMethods[i]).append("\n");
                }
                System.out.println(sb.toString());
            }
```

```
            System.out.println("\n6. Public Methods: ");
            Method[] publicMethods = c.getMethods();
            if (localMethods.length == 0) {
                System.out.println("No public methods found!");
            } else {
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < publicMethods.length; i++) {
                    sb.append("\t").append((i + 1)).append(". ").append(publicMethods[i]).append("\
n");
                }
                System.out.println(sb.toString());
            }

            Constructor<?> defaultConstructor = c.getDeclaredConstructor();
            defaultConstructor.setAccessible(true);
            Object employeeInstance = defaultConstructor.newInstance();
            System.out.println("Is employee an enum? " + c.isEnum());
            System.out.println("Is employee an interface? " + c.isInterface());
            System.out.println("Employee Instance: " + employeeInstance.toString());

            Method setSalaryMethod = find(publicMethods, "setSalary");
            if (setSalaryMethod != null) {
                setSalaryMethod.invoke(employeeInstance, 1000.0);
                System.out.println("setSalary() invoked successfully!");
            } else {
                System.out.println("setSalary method not found!");
            }
            Method getSalaryMethod = find(publicMethods, "getSalary");
            if (getSalaryMethod != null) {
                Object salary = getSalaryMethod.invoke(employeeInstance);
                System.out.println("getSalary() returned: " + salary);
            } else {
                System.out.println("getSalary method not found!");
            }
        } catch (Exception e)   {
            System.out.println(e.getMessage());
        }
    }

    private static Method find(Method[] methods, String what) {
        for (Method m: methods) {
            if (m.toString().contains(what)) {
                return m;
            }
        }
        return null;
    }
    public static void main(String[] args) {
        try {
            Lab21Main lab21 = new Lab21Main();
            Class<?> c = Class.forName("Employee");
            lab21.classFun(c);

        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }    }
```

1. Canonical Class Name: Employee

2. Member Fields:

    1. private java.lang.String Employee.firstName
    2. private java.lang.String Employee.lastName
    3. private java.lang.String Employee.id
    4. private double Employee.salary

3. Local Constructors:
    1. public Employee(java.lang.String,java.lang.String,java.lang.String,double)
    2. public Employee()

4. Public Constructors:
    1. public Employee(java.lang.String,java.lang.String,java.lang.String,double)
    2. public Employee()

5. Local Methods:
    1. public java.lang.String Employee.toString()
    2. public java.lang.String Employee.getId()
    3. public java.lang.String Employee.getFirstName()
    4. public void Employee.setFirstName(java.lang.String)
    5. public java.lang.String Employee.getLastName()
    6. public void Employee.setLastName(java.lang.String)
    7. public void Employee.setId(java.lang.String)
    8. public void Employee.giveRaise(double)
    9. public void Employee.setSalary(double)
    10. public double Employee.getSalary()

6. Public Methods:
    1. public java.lang.String Employee.toString()
    2. public java.lang.String Employee.getId()
    3. public java.lang.String Employee.getFirstName()
    4. public void Employee.setFirstName(java.lang.String)
    5. public java.lang.String Employee.getLastName()
    6. public void Employee.setLastName(java.lang.String)
    7. public void Employee.setId(java.lang.String)
    8. public void Employee.giveRaise(double)
    9. public void Employee.setSalary(double)
    10. public double Employee.getSalary()
    11. public boolean java.lang.Object.equals(java.lang.Object)
    12. public native int java.lang.Object.hashCode()
    13. public final native java.lang.Class java.lang.Object.getClass()
    14. public final native void java.lang.Object.notify()
    15. public final native void java.lang.Object.notifyAll()
    16. public final void java.lang.Object.wait(long) throws java.lang.InterruptedException
    17. public final void java.lang.Object.wait(long,int) throws
java.lang.InterruptedException
    18. public final void java.lang.Object.wait() throws java.lang.InterruptedException

Is employee an enum? false
Is employee an interface? false
Employee Instance: 0.00

setSalary() invoked successfully!
getSalary() returned: 1000.0

Lab 22 – Server Client (Unsafe)

CLIENT

```
import java.io.*;
import java.net.Socket;
import java.util.Scanner;

public class Client {
    public static void main(String[] args) {
        Socket clientSocket = null;
        String address = "localhost";
        int port = 8001;
        try {
            if (args.length == 2) {
                address = args[0];
                port = Integer.parseInt(args[1]);
            }
            clientSocket = new Socket(address, port);
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(
                    new BufferedWriter(
                            new OutputStreamWriter(clientSocket.getOutputStream())));
            Scanner scanner = new Scanner(System.in);
            while (true) {
                String outMessage = "Enter a string with your keyboard: ";
                System.out.println(outMessage);

                String inMessage = scanner.nextLine();
                out.println(inMessage);
                out.flush();
                String serverResponse = in.readLine();
                if (serverResponse.equalsIgnoreCase("QUIT")) {
                    break;
                }
                System.out.println("Server: " + serverResponse);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (clientSocket != null) {
                try {
                    clientSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

SERVER

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class Server {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket clientConnection = null;
        int port = 8001;
        try {
            if (args.length == 1) {
                port = Integer.parseInt(args[0]);
            }
            serverSocket = new ServerSocket(port);
            clientConnection = serverSocket.accept();
            handleClient(clientConnection);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (clientConnection != null) {
                try {
                    clientConnection.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public static void handleClient(Socket clientConnection) throws IOException   {
        Scanner in = new Scanner(clientConnection.getInputStream());
        PrintWriter out = new PrintWriter(
                new BufferedWriter(new OutputStreamWriter(clientConnection.getOutputStream())));
        while (true) {
            if (!in.hasNextLine()) {
                break;
            }
            String message = in.nextLine();
            System.out.println("Server received: " + message);
            if (message.equalsIgnoreCase("QUIT")) {
                out.println("QUIT");
                out.flush();
                break;
            }
            out.println("Okay from the server");
            out.flush();
        }
    }
}
```

Lab 23 – Client Server (Safe)

CLIENT

```
import java.net.*; import java.io.*;
import java.util.Scanner;

public class Client {
    public static void main(String args[]) {
        Socket s = null;
        Scanner scanner = new Scanner(System.in);
        try {
            int port = 8001;
            String address = null;
            if (args.length == 2) {
                address = args[0];
                port = Integer.parseInt(args[1]);
            }
            s = new Socket(address, port);
            DataInputStream in = new DataInputStream(s.getInputStream());
            DataOutputStream out = new DataOutputStream(s.getOutputStream());

            String message;
            while (true) {
                System.out.print("Enter message: ");
                message = scanner.nextLine();
                // void writeUTF(String)
                out.writeUTF(message);
                if (message.toUpperCase().equals("QUIT")) break;
                // String readUTF()
                String data = in.readUTF();
                System.out.println("Received: " + data);
            }
        } catch (UnknownHostException e) {
            System.out.println("Sock:" + e.getMessage());
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO:" + e.getMessage());
        } finally {
            if (s != null)
                try {
                    s.close();
                } catch (IOException e) {
                    System.out.println("client close() failed");
                }
        }
    }
}
```

SERVER

```
//TCP server makes a connection for each client and then echoes the client's request
import java.net.*; import java.io.*;
public class Server {
    public static void main (String args[]) {
        try {
            int port = 8001;
            if (args.length == 1) {
                port = Integer.parseInt(args[0]);
            }
            ServerSocket listenSocket = new ServerSocket(port);
            while(true) {
                Socket clientSocket = listenSocket.accept();
                ServerWorker c = new ServerWorker(clientSocket);
            }
        } catch(IOException e) {
            System.out.println("Listen :"+e.getMessage());
        }
    }
}

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

public class ServerWorker extends Thread{
    DataInputStream dataInputStream;
    DataOutputStream dataOutputStream;
    Socket socket;
    Scanner scanner;
    static int clientCounter = 0;

    ServerWorker (Socket socket)   {
        this.socket = socket;
        try {
            this.dataInputStream = new DataInputStream(socket.getInputStream());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        try {
            this.dataOutputStream = new DataOutputStream(socket.getOutputStream());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        this.scanner = new Scanner(System.in);
        this.start();
    }

    public void run() {
        clientCounter++;
        String message;
        String reply;
        System.out.println("Starting new connection for " + clientCounter);

        try {
            while (true) {
                try {
```

```
                message = dataInputStream.readUTF();
                if (message.equals("QUIT")) {
                    break;
                }
                System.out.println(clientCounter + ") Server received: " + message);
                System.out.println(clientCounter + ") Enter a reply: ");
                reply = scanner.nextLine();
                dataOutputStream.writeUTF(reply);
            } catch (EOFException e) {
                System.out.println("End of input stream reached");
                break;
            }
        }
    } catch (IOException e) {
        System.out.println("IO error occurred: " + e.getMessage());
    } finally {
        try {
            if (socket != null) {
                socket.close();
            }
        } catch (IOException e) {
            System.out.println("Error closing socket: " + e.getMessage());
        }
    }
}


}
```

Lab 24 – Threading

```
import java.util.Scanner;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.atomic.AtomicInteger;

public class Lab24Main implements Runnable {

	// Shared queue for Parcel objects
	private BlockingQueue<Parcel> queue;

	// Shared Stats object
	private Stats stats;

	// Shared counter
	public static AtomicInteger counter;

   // Robots and threads
	private Robot[] robots;
	private Thread[] threads;

	private static Scanner scanner = new Scanner(System.in);
	private int numbots;
	private long beginTime, endTime;

   // Sets up the problem
   // Don't change this code
	public static void main(String[] args) {
		Lab24Main lab = new Lab24Main();

		System.out.print("Enter the number of robots: ");
		int count = scanner.nextInt();

      // Header for robot output
      System.out.println("\nPackage Delivery Times, per robot\n");
		for (int i=0; i<count; i++) {
			System.out.print("Robot#" + i + "\t\t\t");
		}
		System.out.println("\n-------------------------------------------------");

      // Call setup to create the shared data structures,
      // the robots, and start the robot threads
		lab.setup(count);

      // Create the warehouse thread and run it
		Thread warehouse = new Thread(lab);
		warehouse.start();
	}

   // Creates the data structures, robots, and threads
   // Starts the threads
	private void setup(int count) {

      // New up the data structures
		numbots = count;
		queue = new LinkedBlockingQueue<>();
		stats = new Stats(numbots);
		counter = new AtomicInteger(numbots);
```

```
            // Create robot and thread arrays
            robots = new Robot[numbots];
            threads = new Thread[numbots];

            for (int i=0; i< numbots; i++) {
    // YOUR CODE HERE
    // In this loop, for each robot:
    // 1. new up a robot with the
    //    shared data structures
     robots[i] = new Robot(queue, stats);

    // 2. new up a thread with a robot
     threads[i] = new Thread(robots[i]);

               // 3. start the thread
     threads[i].start();

            }
   }

   @Override
   public void run() {
// This is the warehouse code
// The warehouse "receives" packages by
//    new'ing them up; then it puts them
//    on the shared queue; then it sleeps for
//    5 time units to slow it down a little;
//    continue while the shared counter is positive
//    - the robots will decrement the counter, not
//    the warehouse.
//  At the end, it should wait for the robot
//    threads to finish their work
            beginTime = System.currentTimeMillis();
// Loop while the counter is positive
while (counter.get() > 0) {
    // YOUR CODE HERE
    // Create a Parcel
                    Parcel parcel = new Parcel();
    // offer it to the queue
                    queue.offer(parcel);
    // Sleep for 5 units
    try {
                            Thread.sleep(5);
                    } catch (InterruptedException e) {
                            e.printStackTrace();
                    }
}
// Loop to join the threads
// YOUR CODE HERE
            for (Thread t: threads) {
    try {
        t.join();
    } catch (InterruptedException e) {
        System.out.println("Error: " + e.getMessage());
    }
}

            endTime = System.currentTimeMillis();
            printStats();
   }
```

```
    // Print the final simulation statistics
    // Don't change this code
        public void printStats() {
                System.out.println("\n\nRobot Summary");
                System.out.format("%9s %10s %15s %15s\n", "Robot#", "# parcels", "Parcel
Time", "Running Time");
                int totalParcels = 0;
                int totalTime = 0;
                long totalRunningTime = 0;
                for (int i=0; i<numbots; i++) {
                        totalParcels += stats.getParcel(i);
                        totalTime += stats.getTimes(i);
                        totalRunningTime += stats.getRobotTime(i);
                        System.out.format("%9d %10d %15d %15d\n", i, stats.getParcel(i),
stats.getTimes(i), stats.getRobotTime(i));
                }
                System.out.println("-----------------------------------------------------");
                System.out.format("%9s %10d %15d %15d\n",
                                "Totals: "+numbots, totalParcels, totalTime, totalRunningTime);
                System.out.println("\nTotal elapsed time: " + (endTime-beginTime));
                System.out.println("Number of parcels left on queue: " + queue.size());
        }

}

import java.util.Random;

// Don't change this code
public class Parcel {
        private int id;
        private int deliveryTime;
        private static int idCount = 0;
        private Random random = new Random();

        public Parcel() {
                id = idCount++;
                deliveryTime = random.nextInt(20) + 5;
        }

        public int getDeliveryTime() {
                return deliveryTime;
        }

        public int getId() {
                return id;
        }

}

import java.util.concurrent.BlockingQueue;

public class Robot implements Runnable {
        private int id;
        private static int idCount = 0;
        private long beginTime, endTime;
        private int battery = 100;   // Use this for the loop condition
        private BlockingQueue<Parcel> queue;
        private Stats stats;

        public Robot(BlockingQueue<Parcel> queue, Stats stats) {
                this.queue = queue;
```

```
                this.stats = stats;
                id = idCount++;
        }

        @Override
        public void run() {
                beginTime = System.currentTimeMillis();
                Parcel p = null;

      // Loop until the battery dies
       while (battery > 0) {
          // YOUR CODE HERE
          // take() a parcel
          try {
             p = queue.take();
          } catch (InterruptedException e) {
                                System.out.println("Error: " + e.getMessage());
          }
          // Thread.sleep() for the parcel's delivery time
                        try {
                                Thread.sleep(p.getDeliveryTime());
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
          // decrement the battery
                        battery -= p.getDeliveryTime();
          // print parcel data in a synchronized block
          synchronized (stats) {
             // This is just for spacing
             for(int i=0; i<id; i++) {
                System.out.print("\t\t\t");
             }
             // Delivery time
                                System.out.println("parcel #" + p.getId() + " in " +
p.getDeliveryTime());
          }

          // update the stats for robot #id
          // Don't change these lines
                        stats.putParcel(id);
                        stats.putTime(id, p.getDeliveryTime());
      }

                endTime = System.currentTimeMillis();
                // update robot running time
                stats.putRobotTime(id, endTime-beginTime);
                // update the shared counter
                Lab24Main.counter.getAndDecrement();
        }

}

// Don't change this class

import java.util.Arrays;

public class Stats {
        private int[] parcels;
        private int[] times;
        private long[] robotTimes;
        private int numberOfRobots;
```

```
	public Stats(int numberOfRobots) {
		this.numberOfRobots = numberOfRobots;
		this.parcels = new int[numberOfRobots];
		this.times = new int[numberOfRobots];
		this.robotTimes = new long[numberOfRobots];
	}

	public synchronized void putParcel(int robotNumber) {
		parcels[robotNumber]++;
	}

	public synchronized void putTime(int robotNumber, int time) {
		times[robotNumber] += time;
	}

	public synchronized void putRobotTime(int robotNumber, long time) {
		robotTimes[robotNumber] = time;
	}

	public synchronized int getParcel(int robotNumber) {
		return parcels[robotNumber];
	}

	public synchronized int getTimes(int robotNumber) {
		return times[robotNumber];
	}

	public synchronized long getRobotTime(int robotNumber) {
		return robotTimes[robotNumber];
	}

}
```