CS 1073

FR04B

Assignment 8

Ebrahim Arefi

3621326

Q1 Basic Inheritance

RestaurantWorker:

```
/**
 * This class represents a restaurant worker.
 * Stores the worker’s name, job title, and hourly rate.
 * Computes their total pay based on hours worked.
 *
 * @author Ebrahim Arefi, 3621326
 */
public class RestaurantWorker {

    /**
     * The worker's name.
     */
    private String name;

    /**
     * The worker's job title.
     */
    private String jobTitle;

    /**
     * The worker's hourly pay rate.
     */
    private double hourlyRate;

    /**
     * Constructs a RestaurantWorker with the given name, job title, and
hourly
     * rate.
     *
     * @param nameIn       the worker’s name.
     * @param jobTitleIn   the worker’s job title.
     * @param hourlyRateIn the hourly pay rate.
     */
    public RestaurantWorker(String name, String jobTitle, double
hourlyRate) {
        this.name = name;
        this.jobTitle = jobTitle;
        this.hourlyRate = hourlyRate;
    }

    /**
     * Accessor method that retrieves worker's name.
     *
     * @return the worker’s name.
     */
    public String getName() {
        return name;
    }

    /**
     * Accessor method that retrieves the worker's job title.
     *
     * @return the worker’s job title.
     */
    public String getJobTitle() {
        return jobTitle;
    }

    /**
     * Accessor method that retrieves the worker's hourly rate.
     *
     * @return the worker’s hourly rate.
     */
    public double getHourlyRate() {
        return hourlyRate;
    }

    /**
     * Additional method to compute the salary.
     *
     * @param hours the number of hours worked.
     * @return the total pay.
     */
    public double computePay(double hours) {
        return hourlyRate * hours;
    }
}
```

Server:

```
/**
 * This class represents a server in a restaurant who receives an additional
 * allowance.
 * The footwear allowance is paid only for full hours worked.
 *
 * @author Ebrahim Arefi, 3621326
 */
public class Server extends RestaurantWorker {

    /**
     * The footwear allowance amount per hour.
     */
    private double allowance;

    /**
     * Constructs a Server with the given name, job title, hourly rate, and
     * allowance.
     *
     * @param nameIn       the server’s name.
     * @param jobTitleIn   the server’s job title.
     * @param hourlyRateIn the hourly pay rate.
     * @param allowanceIn  the allowance per hour.
     */
    public Server(String name, String jobTitle, double hourlyRate, double allowance) {
        super(name, jobTitle, hourlyRate);
        this.allowance = allowance;
    }

    /**
     * Accessor method that retrieves the allowance amount.
     *
     * @return the allowance per hour.
     */
    public double getAllowance() {
        return allowance;
    }

    /**
     * Calculates the total weekly pay for the server, considering their allowance.
     *
     * @param hours the number of hours worked.
     * @return the total pay including allowance.
     */
    public double computePay(double hours) {

        double hourlySalary = getHourlyRate() * hours;
        double bonus = (int) hours * allowance;

        return hourlySalary + bonus;
    }
}
```

Chef:

```java
/**
 * This class represents a chef in a restaurant.
 * Calculates total weekly pay after union fees.
 *
 * @author Ebrahim Arefi, 3621326
 */
public class Chef extends RestaurantWorker {
    /**
     * The union fees deducted from pay.
     */
    private double unionFees;

    /**
     * Constructs a Chef with the given name, job title, hourly rate, and union
     * fees.
     *
     * @param nameIn       the chef’s name.
     * @param jobTitleIn   the chef’s job title.
     * @param hourlyRateIn the hourly pay rate.
     * @param unionFeesIn  the union fees.
     */
    public Chef(String name, String jobTitle, double hourlyRate, double unionFees) {
        super(name, jobTitle, hourlyRate);
        this.unionFees = unionFees;
    }

    /**
     * Accessor method that retrieves the union fees.
     *
     * @return the union fees.
     */
    public double getUnionFees() {
        return unionFees;
    }

    /**
     * Calculates the chef’s total weekly pay after union fees considering overtime
     * pay.
     *
     * @param hours the number of hours worked.
     * @return the total weekly pay after deductions.
     */
    public double computePay(double hours) {
        double weeklySalary;

        if (hours <= 40) {
            weeklySalary = hours * getHourlyRate();

        } else {
            double overtimeHours = hours - 40;
            double overtimePay = overtimeHours * (getHourlyRate() * 1.5);
            weeklySalary = (40 * getHourlyRate()) + overtimePay;
        }
        return weeklySalary - unionFees;
    }
}
```

Payroll

```
/**
 * A driver class named to calculate the Payroll of each worker.
 *
 * @author Ebrahim Arefi, 3621326
 */

import java.text.NumberFormat;

public class Payroll {
        public static void main(String[] agrs) {

                NumberFormat currencyFormat = NumberFormat.getCurrencyInstance();

                RestaurantWorker Adriano = new RestaurantWorker("Adriano Oliveira", "Busser",
15.75);
                RestaurantWorker Fiona = new RestaurantWorker("Fiona Grant-Long", "Busser",
15.95);
                RestaurantWorker Hoang = new RestaurantWorker("Hoang Nguyen", "Dishwasher",
16.50);

                Server Jonathan = new Server("Jonathan Gorman", "Server", 16.5, 0.12);
                Server Aisha = new Server("Aisha Adegboyega", "Server", 15.75, 0.10);
                Server Brittany = new Server("Brittany Phillips", "Server", 17.50, 0.15);

                Chef Laura = new Chef("Laura Cox", "Executive Chef", 28.50, 57.00);
                Chef Adeline = new Chef("Adeline Gagne", "Sous Chef", 23.50, 43.50);
                Chef Nathaniel = new Chef("Nathaniel Paul", "Sous Chef", 25.75, 43.50);
                Chef Eleanor = new Chef("Eleanor Ryan", "Pastry Chef", 22.00, 39.00);

                System.out.println("Worker's Name & Job Title\tRate of Pay\tPay this week");
                System.out.println("=========================\t===========\t===============");

                System.out.println(Fiona.getName() + " (" + Fiona.getJobTitle() + ")\t" +
                                currencyFormat.format(Fiona.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Fiona.computePay(14.5)) + "\n--------------
-----------");

                System.out.println(Adriano.getName() + " (" + Adriano.getJobTitle() + ")\t" +
                                currencyFormat.format(Adriano.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Adriano.computePay(18)) + "\n--------------
-----------");

                System.out.println(Aisha.getName() + " (" + Aisha.getJobTitle() + ")\t" +
                                currencyFormat.format(Aisha.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Aisha.computePay(18)) + "\n----------------
---------");

                System.out.println(Adeline.getName() + " (" + Adeline.getJobTitle() + ")\t" +
                                currencyFormat.format(Adeline.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Adeline.computePay(18)) + "\n--------------
-----------");

                System.out.println(Nathaniel.getName() + " (" + Nathaniel.getJobTitle() + ")\t" +
                                currencyFormat.format(Nathaniel.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Nathaniel.computePay(26)) + "\n------------
-------------");

                System.out.println(Brittany.getName() + " (" + Brittany.getJobTitle() + ")\t" +
                                currencyFormat.format(Brittany.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Brittany.computePay(38.5)) + "\n-----------
--------------");

                System.out.println(Hoang.getName() + " (" + Hoang.getJobTitle() + ")\t" +
                                currencyFormat.format(Hoang.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Hoang.computePay(42)) + "\n----------------
---------");

                System.out.println(Eleanor.getName() + " (" + Eleanor.getJobTitle() + ")\t" +
                                currencyFormat.format(Eleanor.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Eleanor.computePay(42)) + "\n--------------
-----------");

                System.out.println(Jonathan.getName() + " (" + Jonathan.getJobTitle() + ")\t" +
                                currencyFormat.format(Jonathan.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Jonathan.computePay(46.5)) + "\n-----------
--------------");

                System.out.println(Laura.getName() + " (" + Laura.getJobTitle() + ")\t" +
                                currencyFormat.format(Laura.getHourlyRate()) + "  /hr\t" +
                                currencyFormat.format(Laura.computePay(46.5)) + "\n--------------
-----------");
        }
}
```

Q1 Output

```
ebi@Ebis-MBP as8 % java Payroll
Worker's Name & Job Title Rate of Pay  Pay this week
=========================    ===========      ===============
Fiona Grant-Long (Busser)    $15.95 /hr       $231.27
-----------------------
Adriano Oliveira (Busser)    $15.75 /hr       $283.50
-------------------------
Aisha Adegboyega (Server)    $15.75 /hr       $285.30
-------------------------
Adeline Gagne (Sous Chef)    $23.50 /hr       $379.50
-------------------------
Nathaniel Paul (Sous Chef)   $25.75 /hr       $626.00
-------------------------
Brittany Phillips (Server)   $17.50 /hr       $679.45
-------------------------
Hoang Nguyen (Dishwasher)    $16.50 /hr       $693.00
-------------------------
Eleanor Ryan (Pastry Chef)   $22.00 /hr       $907.00
-------------------------
Jonathan Gorman (Server)     $16.50 /hr       $772.77
-------------------------
Laura Cox (Executive Chef)   $28.50 /hr       $1,360.88
```

Q2 Conference Pass Sales:

Conference pass:

```
/**
 * This abstract class represents a general conference pass.
 * Stores the participant’s name and ACM membership status.
 *
 * @author Ebrahim Arefi, 3621326
 */
public abstract class ConferencePass {

    /**
     * The participant’s name.
     */
    private String name;

    /**
     * Indicates whether the participant is an ACM member.
     */
    private boolean memberAcm;

    /**
     * Constructs a ConferencePass with the given name and membership status.
     *
     * @param nameIn      the participant’s name.
     * @param memberAcmIn true if the participant is an ACM member.
     */
    public ConferencePass(String nameIn, boolean memberAcmIn) {
        name = nameIn;
        memberAcm = memberAcmIn;
    }

    /**
     * Retrieves the participant’s name.
     *
     * @return the participant’s name.
     */
    public String getName() {
        return name;
    }

    /**
     * Retrieves the ACM membership status.
     *
     * @return true if the participant is an ACM member, false otherwise.
     */
    public boolean getMemberAcm() {
        return memberAcm;
    }

    /**
     * Abstract method to calculate the total cost of the pass.
     *
     * @return the total cost of the conference pass.
     */
    public abstract double totalCost();
}
```

ExhibitorPass

```
/**
 * This class represents an exhibitor pass.
 * Calculates the total cost.
 * Has ACM membership discounts.
 *
 * @author Ebrahim Arefi, 3621326
 */
public class ExhibitorPass extends ConferencePass {

    /**
     * The number of tables requested for the exhibitor’s booth.
     */
    private int numOfTables;

    /**
     * Indicates whether power outlets are required for the booth.
     */
    private boolean powerOutlet;

    /**
     * Constructs an ExhibitorPass with the given information.
     *
     * @param nameIn        the exhibitor’s name.
     * @param memberAcmIn   true if the exhibitor is an ACM member.
     * @param numOfTablesIn the number of tables requested.
     * @param powerOutletIn true if power outlets are required.
     */
    public ExhibitorPass(String nameIn, boolean memberAcmIn, int numOfTablesIn,
boolean powerOutletIn) {
        super(nameIn, memberAcmIn);
        numOfTables = numOfTablesIn;
        powerOutlet = powerOutletIn;
    }

    /**
     * Retrieves the number of tables for the exhibitor’s booth.
     *
     * @return the number of tables.
     */
    public int getNumOfTables() {
        return numOfTables;
    }

    /**
     * Retrieves whether power outlets are required.
     *
     * @return true if power outlets are required, false otherwise.
     */
    public boolean getPowerOutlet() {
        return powerOutlet;
    }
    /**
     * Calculates the total cost of the exhibitor pass.
     * Includes base price, table costs, and applies a 15% discount for ACM members.
     *
     * @return the total cost of the exhibitor pass.
     */
    public double totalCost() {

        double basePrice;
        double tableCost = numOfTables * 9.45;

        if (!powerOutlet) {
            basePrice = 350.75;
        } else {
            basePrice = 420.55;
        }

        double Cost = basePrice + tableCost;

        if (getMemberAcm()) {
            Cost = Cost - (Cost * 0.15);
        }

        return Cost;

    }

}
```

Attendee pass:

```
/**
 * This class represents an attendee pass.
 * Calculates the total cost.
 *
 * @author Ebrahim Arefi, 3621326
 */
public class AttendeePass extends ConferencePass {

	/**
	 * The number of extra banquet tickets purchased for guests.
	 */
	private int numExtraBanquetTickets;

	/**
	 * Constructs an AttendeePass with the given name, ACM membership,
	 * and number of extra banquet tickets.
	 *
	 * @param nameIn                 the attendee’s name.
	 * @param memberAcmIn            true if the attendee is an ACM member.
	 * @param numExtraBanquetTickets the number of additional banquet tickets.
	 */
	public AttendeePass(String nameIn, boolean memberAcmIn, int
numExtraBanquetTickets) {
		super(nameIn, memberAcmIn);
		this.numExtraBanquetTickets = numExtraBanquetTickets;
	}

	/**
	 * Retrieves the total number of banquet tickets
	 * including the attendee’s own ticket.
	 *
	 * @return total number of banquet tickets.
	 */
	public int getTotalBanquetTickets() {
		return 1 + numExtraBanquetTickets;
	}

	/**
	 * Calculates the total cost of the attendee pass.
	 * Has a 20% discount for ACM members.
	 *
	 * @return total cost of the attendee pass.
	 */
	public double totalCost() {
		double total;
		double basePrice = 225.00;
		double selfbanquetTikcetPrice = 45.00;
		double guestBanquetTicketPrice = 53.25;

		total = basePrice + selfbanquetTikcetPrice; // attendee Ticket Price only
		double guestPrice = numExtraBanquetTickets * guestBanquetTicketPrice; //
guest price = number of their tickets *
		// specific price for each extra guest.

		total += guestPrice; // total = attendee + guest

		if (getMemberAcm()) {
			total = total - (total * 0.20);
		}
		return total;
	}

}
```

Student Attendee Pass

```
import java.util.Random;

/**
 * This class represents a student attendee pass.
 * Calculates total cost after applying a sponsor discount
 * and assigns a random giveaway item to the student.
 *
 * @author Ebrahim Arefi, 3621326
 */
public class StudentAttendeePass extends AttendeePass {

	/**
	 * The randomly selected giveaway item for the student.
	 */
	private String giveawayItem;

	/**
	 * Constructs a StudentAttendeePass with the given name, membership status,
	 * and number of extra banquet tickets. A random giveaway item is assigned
	 * upon creation.
	 *
	 * @param nameIn                 the student’s name.
	 * @param memberAcmIn            true if the student is an ACM member.
	 * @param extraBanquetTicketsIn the number of extra banquet tickets purchased.
	 */
	public StudentAttendeePass(String nameIn, boolean memberAcmIn, int 
extraBanquetTicketsIn) {
		super(nameIn, memberAcmIn, extraBanquetTicketsIn);

		Random random = new Random();
		int item = random.nextInt(4);

		if (item == 0) {
			giveawayItem = "T-shirt";
		} else if (item == 1) {
			giveawayItem = "Water bottle";
		} else if (item == 2) {
			giveawayItem = "Travel mug";
		} else {
			giveawayItem = "Lanyard";
		}
	}

	/**
	 * Retrieves the giveaway item assigned to the student.
	 *
	 * @return the giveaway item.
	 */
	public String getGiveawayItem() {
		return giveawayItem;
```

```java
        }

        /**
         * Calculates the total cost of the student attendee pass.
         * Has a $150 sponsor discount to the total cost.
         *
         * @return the total cost after the sponsor discount.
         */
        public double totalCost() {
              double cost = super.totalCost() - 150.00;
              return cost;
        }
}
```

ConferenceDriver

```java
import java.text.NumberFormat;

/**
 * This class tests the ConferencePass subclasses.
 * It creates multiple types of passes and prints their details and total costs.
 *
 * @author Ebrahim Arefi, 3621326
 */
public class ConferenceDriver {

    public static void main(String[] args) {

        NumberFormat currencyFormat = NumberFormat.getCurrencyInstance();

        ExhibitorPass e1 = new ExhibitorPass("Ebi A", true, 3, true);
        ExhibitorPass e2 = new ExhibitorPass("King Kong", false, 1, false);

        AttendeePass a1 = new AttendeePass("Kim Kar", true, 2);
        AttendeePass a2 = new AttendeePass("Jacki Lee", false, 0);

        StudentAttendeePass s1 = new StudentAttendeePass("Lee Chan", true, 12);
        StudentAttendeePass s2 = new StudentAttendeePass("Meo Dog", false, 1);

        System.out.println("\n     Exhibitor Pass Details");
        System.out.println("================================\n");

        System.out.println("Pass Type: Exhibitor Pass");
        System.out.println("Name: " + e1.getName());
        System.out.println("ACM Member: " + e1.getMemberAcm());
        System.out.println("Number of Tables: " + e1.getNumOfTables());
        System.out.println("Power Outlet Required: " + e1.getPowerOutlet());
        System.out.println("Total Cost: " + currencyFormat.format(e1.totalCost()));
        System.out.println();

        System.out.println("Pass Type: Exhibitor Pass");
        System.out.println("Name: " + e2.getName());
        System.out.println("ACM Member: " + e2.getMemberAcm());
        System.out.println("Number of Tables: " + e2.getNumOfTables());
        System.out.println("Power Outlet Required: " + e2.getPowerOutlet());
        System.out.println("Total Cost: " + currencyFormat.format(e2.totalCost()));
        System.out.println();

        System.out.println("     Attendee Pass Details");
        System.out.println("================================\n");

        System.out.println("Pass Type: Attendee Pass");
        System.out.println("Name: " + a1.getName());
        System.out.println("ACM Member: " + a1.getMemberAcm());
        System.out.println("Total Banquet Tickets: " + a1.getTotalBanquetTickets());
        System.out.println("Total Cost: " + currencyFormat.format(a1.totalCost()));
        System.out.println();

        System.out.println("Pass Type: Attendee Pass");
        System.out.println("Name: " + a2.getName());
        System.out.println("ACM Member: " + a2.getMemberAcm());
        System.out.println("Total Banquet Tickets: " + a2.getTotalBanquetTickets());
        System.out.println("Total Cost: " + currencyFormat.format(a2.totalCost()));
        System.out.println();

        System.out.println("     Student Attendee Pass Details");
        System.out.println("====================================\n");

        System.out.println("Pass Type: Student Attendee Pass");
        System.out.println("Name: " + s1.getName());
        System.out.println("ACM Member: " + s1.getMemberAcm());
        System.out.println("Total Banquet Tickets: " + s1.getTotalBanquetTickets());
        System.out.println("Giveaway Item: " + s1.getGiveawayItem());
        System.out.println("Total Cost: " + currencyFormat.format(s1.totalCost()));
        System.out.println();

        System.out.println("Pass Type: Student Attendee Pass");
        System.out.println("Name: " + s2.getName());
        System.out.println("ACM Member: " + s2.getMemberAcm());
        System.out.println("Total Banquet Tickets: " + s2.getTotalBanquetTickets());
        System.out.println("Giveaway Item: " + s2.getGiveawayItem());
        System.out.println("Total Cost: " + currencyFormat.format(s2.totalCost()));
    }

}
```

Q2 Output

```
ebi@Ebis-MBP as8 % java ConferenceDriver

     Exhibitor Pass Details
==============================

Pass Type: Exhibitor Pass
Name: Ebi A
ACM Member: true
Number of Tables: 3
Power Outlet Required: true
Total Cost: $381.57

Pass Type: Exhibitor Pass
Name: King Kong
ACM Member: false
Number of Tables: 1
Power Outlet Required: false
Total Cost: $360.20

     Attendee Pass Details
==============================

Pass Type: Attendee Pass
Name: Kim Kar
ACM Member: true
Total Banquet Tickets: 3
Total Cost: $301.20

Pass Type: Attendee Pass
Name: Jacki Lee
ACM Member: false
Total Banquet Tickets: 1
Total Cost: $270.00

    Student Attendee Pass Details
===================================

Pass Type: Student Attendee Pass
Name: Lee Chan
ACM Member: true
Total Banquet Tickets: 13
Giveaway Item: Water bottle
Total Cost: $577.20

Pass Type: Student Attendee Pass
Name: Meo Dog
```

```
ACM Member: false
Total Banquet Tickets: 2
Giveaway Item: Water bottle
Total Cost: $173.25
ebi@Ebis-MBP as8 %
```