



JAVA BASIC

Lab Guides

Lab Guide: Polymorphism & Abstraction

Lab Guide 1: Overload

Specifications

Create a new package named **exercise.methodoverloading** in **TaxManagement** project.

- Create a **BasicRateTax** class with a method **calcTax()** that returns 20% of a fixed base income of £1000.
- Create a java program named **TaxCollector** that creates a new **BasicRateTax** object, calls the **calcTax()** method and prints the output to the console.
 - Run the **TaxCollector** program and ensure it always prints 200.00 as calculated tax.
- Add new **calcTax()** method to **BasicRateTax** class that takes a double *grossIncome* parameter and calculates the tax as 20% of the *grossIncome* if it's greater than the base income of £1000
- Change the **TaxCollector** program to call the new **calcTax(double grossIncome)** method and passing the gross Income value from the command line.
 - Run the **TaxCollector** program and see if the tax is correctly calculated.
 - Re-run the program with different Gross Income values and check the output.

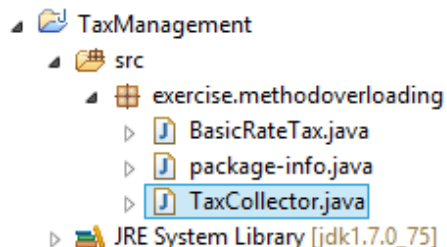
Functional:

Requirement: write a java console program.

- a. How many ways to overload a method? List out invalid case of method overloading?
- b. Question 1: Explain in detail the meaning of code line 9, 10 in **BasicRateTax** class?
- c. Question 2: Describe about piece of code **Double.parseDouble(args[0])**, what is it used for?

Source code

Project struture:



- **BasicRateTax** class:

```
BasicRateTax.java
1 package exercise.methodoverloading;
2
3 /**
4  *
5  * @author DieuNT1
6  *
7  */
8 public class BasicRateTax {
9     private static final double BASE_INCOME = 1000.00;
10    private static final double BASIC_TAX_RATE = 0.20;
11
12    /**
13     * This method calculates fixed base income.
14     *
15     * @return returns 20% of a fixed base income of £1000.
16     */
17    public double calcTax() {
18        return BASE_INCOME * BASIC_TAX_RATE;
19    }
20
21    /**
22     * calculates the tax as 20% of the grossIncome.
23     *
24     * @param grossIncome
25     * @return returns 20% of the grossIncome if it's greater than the base income
26     *         of £1000.
27     */
28    public double calcTax(double grossIncome) {
29        if (grossIncome < BASE_INCOME) {
30            return calcTax();
31        }
32        return grossIncome * BASIC_TAX_RATE;
33    }
34 }
35
```

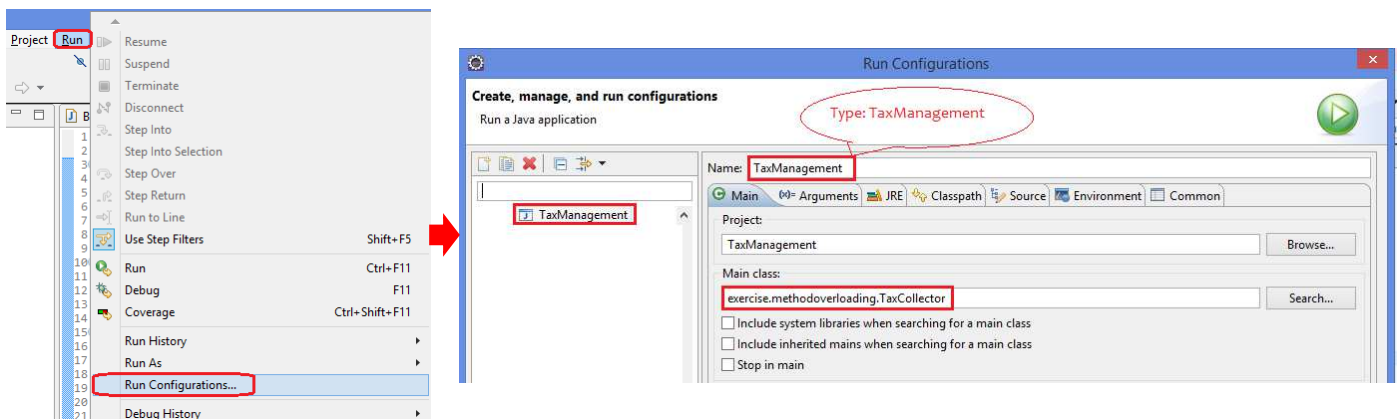
- TaxCollector class:

```
BasicRateTax.java TaxCollector.java
1 package exercise.methodoverloading;
2
3 /**
4  * This class contains the main method to run app.
5  *
6  * @author DieuNT1
7  *
8  */
9 public class TaxCollector {
10    /**
11     * The main method.
12     *
13     * @param args
14     */
15    public static void main(String[] args) {
16        /*
17         * The grossIncome value gets from the first argument of main.
18         */
19        double grossIncome = Double.parseDouble(args[0]);
20        /*
21         * Create a new BasicRateTax object named: taxCalculator.
22         */
23        BasicRateTax taxCalculator = new BasicRateTax();
24    }
25 }
```

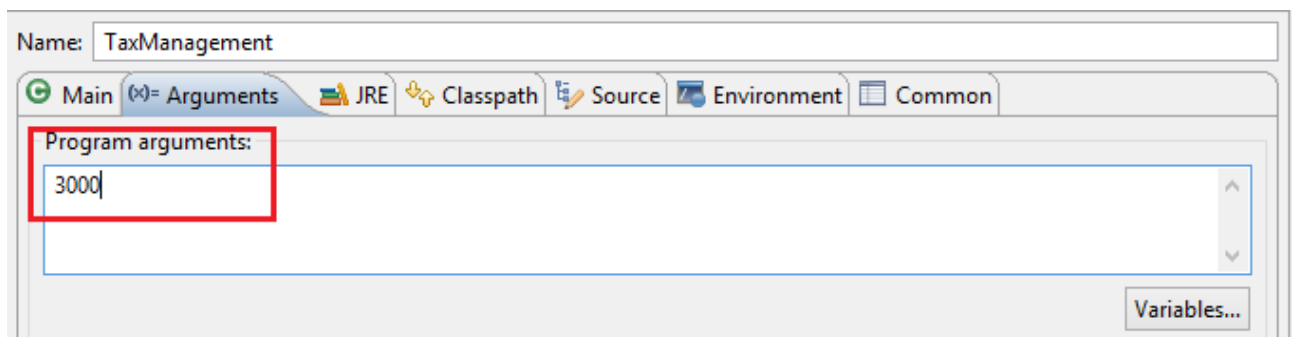
```
24      /*  
25       * Call object's calcTax method.  
26       */  
27      double tax = taxCalculator.calcTax(grossIncome);  
28      |  
29      /*  
30       * Print out result.  
31       */  
32      System.out.println("Tax due is " + tax);  
33  }  
34  }
```

- **How to run:**

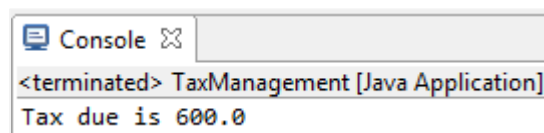
Click **Run** menu | choose **Run Configurations**:



Enter *grossIncome* value at **Program arguments** (example: 3000) | click **Run** button:



Result:



Lab Guide 2: Override

Specifications

- Create a new package named `exercise.inheritance`.
 - Create a class named `HigherRateTax` in the `exercise.inheritance` package that extends `BasicRateTax` and add an empty `calcTax(double grossIncome)` method.
 - Override `HigherRateTax.calcTax(double grossIncome)` method to calculate the tax as follows:
 - 20% of *grossIncome* if up to £34,000 (hint: reuse the `BasicRateTax.calcTax(double grossIncome)` method)
 - 40% of *grossIncome* if above £34,000 but less than £150,000
 - 50% of *grossIncome* if £150,000 or above
 - Run the existing `TaxCollector` program with some large gross income amounts and observe that your changes didn't have any effect on the calculate tax. Why?
 - Change the code of the `TaxCollector` to instantiate `HigherRateTax` instead of `BasicRateTax`.
 - Run the `TaxCollector` program again and observe that now the new percentage is properly applied. You are now using the overridden version of the method `calcTax()`.

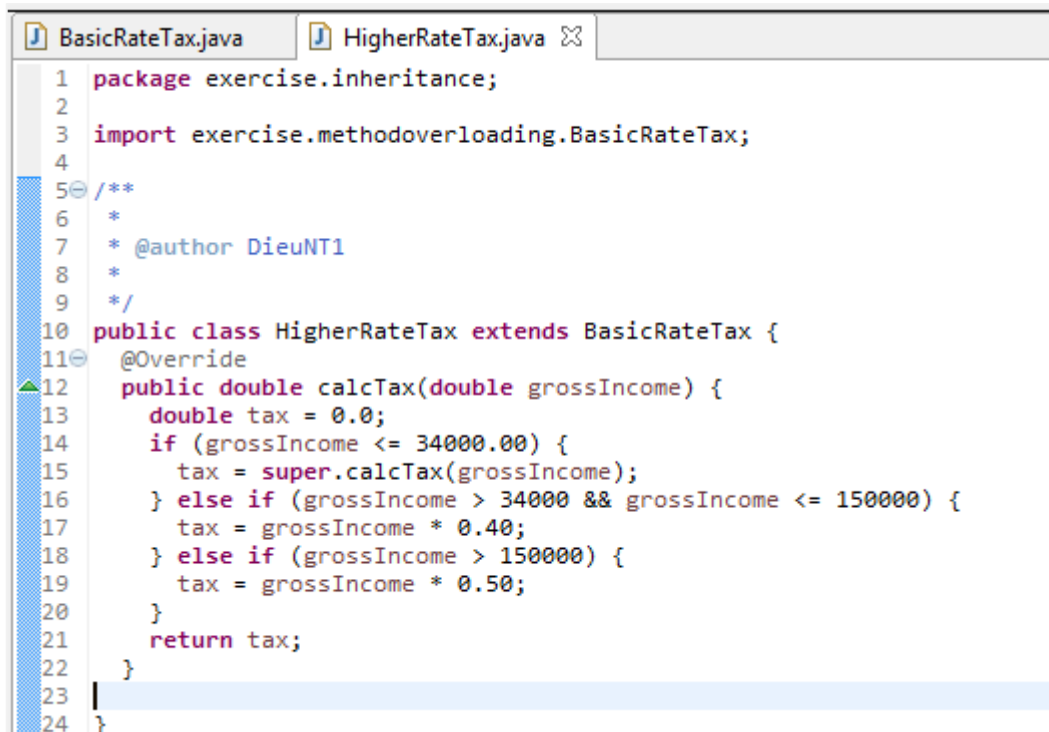
Functional:

Requirement: write a java console program.

- a. Question 2: Describe about piece of code `super.calcTax(grossIncome)`, what is it used for?

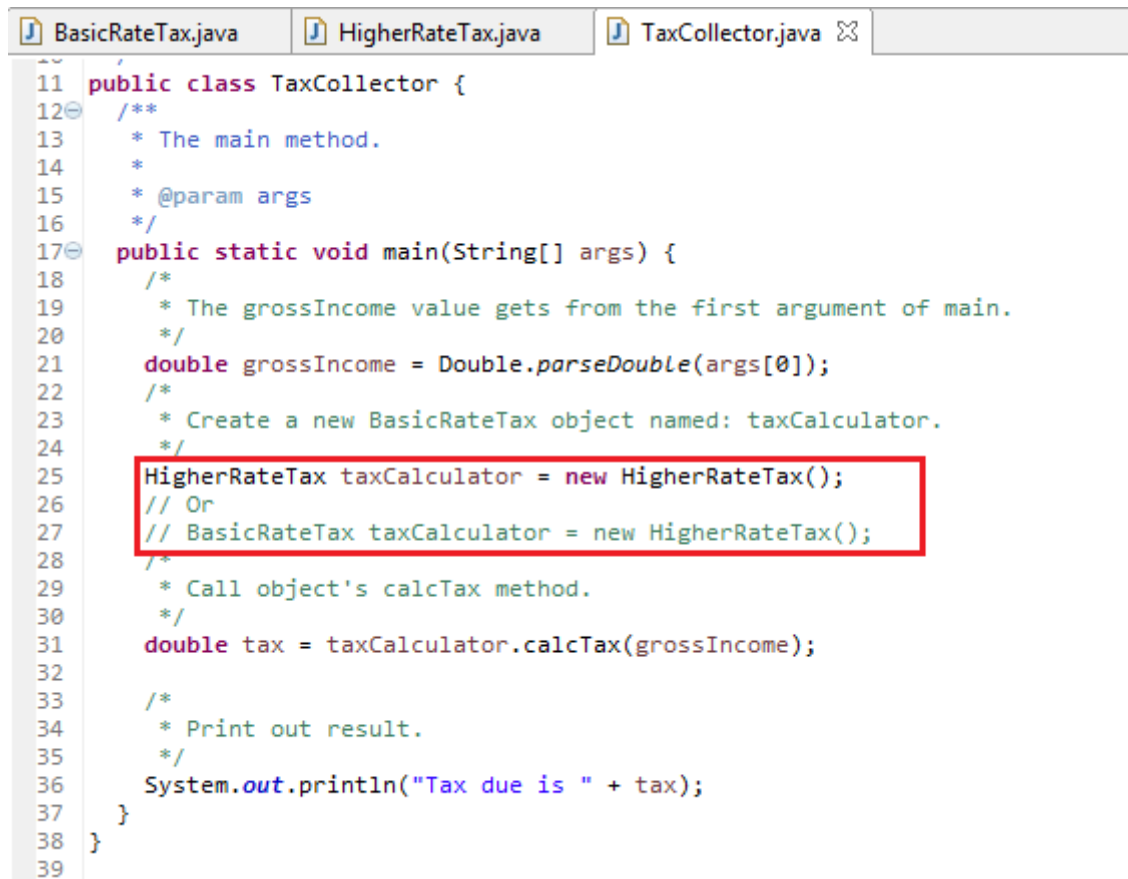
Source code

- `HigherRateTax` class:

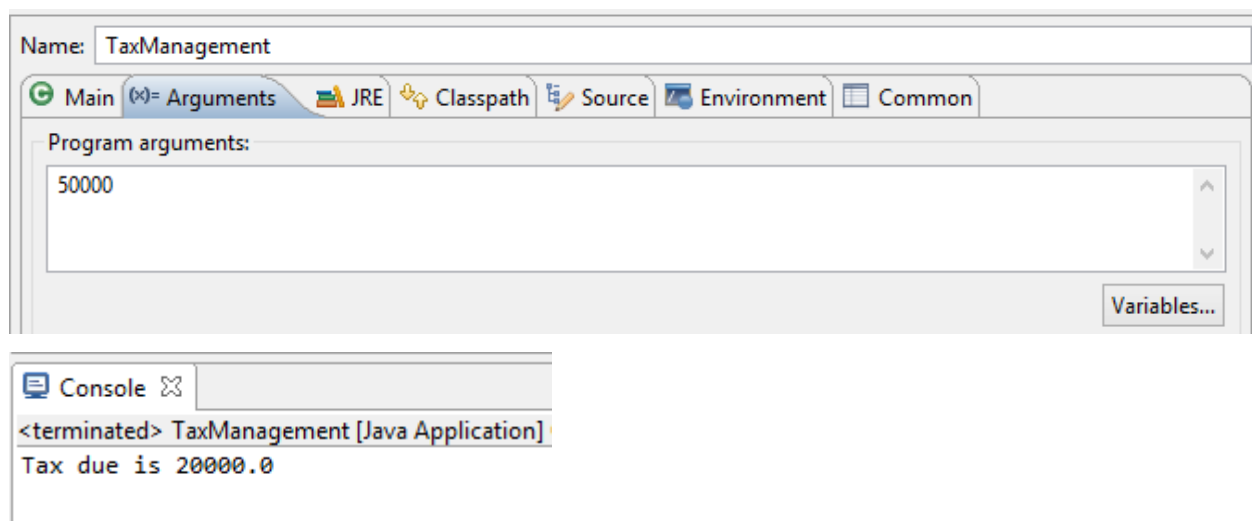


```
1 package exercise.inheritance;
2
3 import exercise.methodoverloading.BasicRateTax;
4
5 /**
6  *
7  * @author DieuNT1
8  *
9  */
10 public class HigherRateTax extends BasicRateTax {
11     @Override
12     public double calcTax(double grossIncome) {
13         double tax = 0.0;
14         if (grossIncome <= 34000.00) {
15             tax = super.calcTax(grossIncome);
16         } else if (grossIncome > 34000 && grossIncome <= 150000) {
17             tax = grossIncome * 0.40;
18         } else if (grossIncome > 150000) {
19             tax = grossIncome * 0.50;
20         }
21         return tax;
22     }
23 }
24 }
```

- `TaxCollector` class: change the code of the `TaxCollector` to instantiate `HigherRateTax` instead of `BasicRateTax`.



```
11 public class TaxCollector {
12     /**
13      * The main method.
14      *
15      * @param args
16      */
17     public static void main(String[] args) {
18         /**
19          * The grossIncome value gets from the first argument of main.
20          */
21         double grossIncome = Double.parseDouble(args[0]);
22         /**
23          * Create a new BasicRateTax object named: taxCalculator.
24          */
25         HigherRateTax taxCalculator = new HigherRateTax();
26         // Or
27         // BasicRateTax taxCalculator = new HigherRateTax();
28         /**
29          * Call object's calcTax method.
30          */
31         double tax = taxCalculator.calcTax(grossIncome);
32
33         /**
34          * Print out result.
35          */
36         System.out.println("Tax due is " + tax);
37     }
38 }
39
```

Result:

Lab Guide 3: Abstract class, Inheritance, Polymorphism

Create a new package named **exercise.abstraction** in EmployeeManagement project.

- Create an abstract class **Employee** with: *employee name, date of birth, address*, **calcSalary()** abstract method, **inputData()** method to input employee information, **display()** method to show all information. This class contains constructor and getter/setter if needs.
- There are three employee type:
 - **Production staff**: amount of product, **salary = amount of product * 20\$**;
 - **Daily staff**: number of workdays, **salary = number of workdays * 15\$**;
 - **Manager**: basic salary, wage, **salary = basic salary * wage**;
- Create class **ProductionStaff**, **DailyStaff** and **Manager** that extends **Employee** class. Override **calcSalary** to calculate salary for each employee type.
- Create another class named **EmployeeManagement** that creates an array contains 3 employees of above type, loop to calls the **display()** method to print the outputs.

Screen Design:

Screen 1: input data

```
Console X
<terminated> EmployeeManagement [Java]
Employee 1
Enter name:
Thanh
Enter birth date:
01/01/1990
Address:
Ha noi
Enter amount of product:
2000
-----
Employee 2
Enter name:
Long
Enter birth date:
12/05/1990
Address:
HCM
Enter numbe of workday:
23
-----
Employee 3
Enter name:
Quang
Enter birth date:
18/06/1992
Address:
Da nang
Enter wage:
100
Enter basic salary:
4000
|
```

Screen 2: display data

```
-----
Thanh    01/01/1990    Ha noi    2000.0    40000.0
Long     12/05/1990    HCM       23.0      345.0
Quang    18/06/1992    Da nang  100.0     4000.0    400000.0|
```

Functional

Requirement: You change the program by adds validation methods to check input data before assign to objects. Demo with n employees.

Source code

✓ Employee class

```
Employee.java
1 package exercise.abstraction;
2
3 import java.util.Scanner;
4
5 public abstract class Employee {
6     private String employeeName;
7     private String dateOfBirth;
8     private String address;
9
10    public abstract double calcSalary();
11
12    /**
13     * User enter data.
14     *
15     * @param scanner
16     */
17    protected void inputData(Scanner scanner) {
18
19        System.out.println("Enter name: ");
20        employeeName = scanner.nextLine();
21
22        System.out.println("Enter birth date: ");
23        dateOfBirth = scanner.nextLine();
24
25        System.out.println("Address: ");
26        address = scanner.nextLine();
27    }
28
29    /**
30     * Display data to console.
31     */
32    protected void display() {
33        System.out.print(employeeName + "\t" + dateOfBirth + "\t" + address);
34    }
35    // getter/setter method
36
```

✓ ProductionStaff class

```
Employee.java  ProductionStaff.java
1 package exercise.abstraction;
2
3 import java.util.Scanner;
4
5 public class ProductionStaff extends Employee {
6
7     private static final int UNIT_PRICE = 20;
8
9     private double amountOfProduct;
10
11    @Override
12    public double calcSalary() {
13        return amountOfProduct * UNIT_PRICE;
14    }
15
```



```
16 @Override
17 protected void inputData(Scanner scanner) {
18     /*
19      * Call inputData method from parent class.
20      */
21     super.inputData(scanner);
22
23     System.out.println("Enter amount of product: ");
24     amountOfProduct = Double.parseDouble(scanner.nextLine());
25
26     System.out.println("-----");
27 }
28
29 @Override
30 protected void display() {
31     // Call method of parent class
32     super.display();
33
34     System.out.print("\t" + amountOfProduct + "\t" + this.calcSalary() + "\n");
35 }
36
37 }
```

✓ **DailyStaff class**

```
Employee.java ProductionStaff.java DailyStaff.java ✕
1 package exercise.abstraction;
2
3 import java.util.Scanner;
4
5 public class DailyStaff extends Employee {
6     private static final int WAGE_DAY = 15;
7
8     private double numbeOfWorkday;
9
10    @Override
11    public double calcSalary() {
12        return numbeOfWorkday * WAGE_DAY;
13    }
14
15    @Override
16    protected void inputData(Scanner scanner) {
17        /*
18         * Call inputData method from parent class.
19         */
20        super.inputData(scanner);
21
22        System.out.println("Enter numbe of workday: ");
23        numbeOfWorkday = Double.parseDouble(scanner.nextLine());
24
25        System.out.println("-----");
26    }
27
28
29    @Override
30    protected void display() {
31        // Call method of parent class
32        super.display();
33
34        System.out.print("\t" + numbeOfWorkday + "\t" + this.calcSalary() + "\n");
35    }
36
37 }
```

✓ **Manager class**

```
Employee.java ProductionStaff.java DailyStaff.java Manager.java ✕
1 package exercise.abstraction;
2
3 import java.util.Scanner;
4
5 public class Manager extends Employee {
6     private double wage;
7     private double basicSalary;
8
9     @Override
10    public double calcSalary() {
11        return wage * basicSalary;
12    }
13
14    @Override
15    protected void inputData(Scanner scanner) {
16        /*
17         * Call inputData method from parent class.
18         */
19        super.inputData(scanner);
20
21        System.out.println("Enter wage: ");
22        wage = Double.parseDouble(scanner.nextLine());
23
24        System.out.println("Enter basic salary: ");
25        basicSalary = Double.parseDouble(scanner.nextLine());
26
27        System.out.println("-----");
28    }
29
30    @Override
31    protected void display() {
32        // Call method of parent class
33        super.display();
34
35        System.out.print("\t" + wage + "\t" + basicSalary + "\t" + this.calcSalary() + "\n");
36    }
37 }
38
```

✓ **EmployeeManagement class**

```
Employee.java EmployeeManagement.java ✕
4
5 public class EmployeeManagement {
6
7     public static void main(String[] args) {
8
9         Employee employees[] = new Employee[3];
10
11         // Create 3 objects
12         ProductionStaff productionStaff = new ProductionStaff();
13         DailyStaff dailyStaff = new DailyStaff();
14         Manager manager = new Manager();
15
16         Scanner scanner = new Scanner(System.in);
17
18         // Call inputData method
19         System.out.println("Employee 1");
20         productionStaff.inputData(scanner);
21
22         System.out.println("Employee 2");
23         dailyStaff.inputData(scanner);
24
25         System.out.println("Employee 3");
26         manager.inputData(scanner);
27     }
28 }
```

```
28 // Push to existed array
29 employees[0] = productionStaff;
30 employees[1] = dailyStaff;
31 employees[2] = manager;
32
33 // loop
34 for (Employee employee : employees) {
35     // An instance of Polymorphims
36     employee.display();
37 }
38
39 scanner.close();
40 }
41 }
```

Lab Guide 4: Static attribute/methods

Thay đổi “Lab Guide 3”, bổ sung thuộc tính tĩnh vào lớp **Employee** như sau:

```
Employee.java
1 package exercise.abstraction;
2
3 import java.util.Scanner;
4
5 public abstract class Employee {
6     private String employeeName;
7     private String dateOfBirth;
8     private String address;
9     private static String companyName;
10
11     public static String getCompanyName() {
12         return companyName;
13     }
14
15     public static void setCompanyName(String companyName) {
16         Employee.companyName = companyName;
17     }
18
19     /**
20      * Display data to console.
21      */
22     protected void display() {
23         System.out.print(employeeName + "\t" + dateOfBirth + "\t" + address + "\t" + companyName);
24     }
25 }
```

Thêm đoạn code sau đây vào lớp EmployeeManagement:

```
Employee.setCompanyName("R2S");
```

Chạy lại chương trình quan sát kết quả và giải thích thay đổi?

Bài tập 1: Xây dựng chương trình quản lý nhân viên cho một công ty. Đối tượng quản lý bao gồm nhân viên phòng **Marketing**, và phòng **Hành chính**. Dựa vào một số đặc tính của từng đối tượng, người quản lý cần đưa ra cách thức đánh giá khác nhau. Hãy xây dựng các lớp sau

	Modifier	Class: Staff
Properties	protected	Tên, mã số, lương căn bản, hệ số lương, lương thực lãnh, thâm niên
Constructors	public	4 tham số: Tên, mã số, lương căn bản, thâm niên
Getters (Thuộc tính chỉ đọc)	public	Lương thực lãnh. Hệ số lương: Mặc định là 1. Cứ mỗi 5 năm công tác, hệ số lương tăng 1 bậc.
Setter (Thuộc tính chỉ đọc)	public	Thâm niên công tác: Không chấp nhận số âm.
Getters & Setter (Thuộc tính đọc và ghi)	public	Tên, mã số, lương căn bản
Methods	public	tinhLuong: tính Lương thực lãnh = lương căn bản * hệ số. Phương thức không trả về kết quả.
		xuatThongTinNV: xuất thông tin nhân viên.

	Modifier	Class: Marketing extends Staff
Properties	protected	Doanh số, hoa hồng.
Constructors	public	5 tham số: Tên, mã số, lương căn bản, thâm niên, doanh số.
Getters (Thuộc tính chỉ đọc)	public	Hoa hồng: Doanh số từ 5.000.000 đến dưới 10tr thì hoa hồng 5% trên doanh số. (Dưới 5tr không có hoa hồng) Doanh số từ 10.000.000 đến dưới 20tr thì hoa hồng 10% trên doanh số. Từ 20.000.000 thì hoa hồng 20% trên doanh số.
Getters & Setter (Thuộc tính đọc và ghi)	public	Doanh số: (setter) không chấp nhận số âm.
Methods	public	tinhLuong: override tính Lương thực lãnh = lương căn bản * hệ số + doanh số * hoa hồng. Hàm không trả về kết quả.
		xuatThongTinNV: override xuất thông tin nhân viên.

	Modifier	Class: Administration extends Staff
Properties	protected	Phụ cấp
Constructors	public	5 tham số: Tên, mã số, lương căn bản, thâm niên, phụ cấp
Getters & Setter	public	Phụ cấp: (setter) không chấp nhận số âm.
Methods	public	tínhLuong: override tính Lương thực lãnh = lương căn bản * hệ số + phụ cấp, không trả về kết quả. xuatThongTinNV: override xuất thông tin nhân viên.

Xây dựng lớp quản lý

	Modifier	Class: StaffManager
Properties	private	Mảng các Staff, số nhân viên
Constructors	public	0 tham số: Cho tối đa mảng có 100 staff, số nhân viên =0;
Methods	public	nhapDS: Nhập danh sách các nhân viên (có thể là nhân viên marketing hoặc nhân viên hành chính) xuatDS: Nhập danh sách các nhân viên maxLuong: Tìm người có lương cao nhất tinhLuongTB: Tính lương trung bình toàn công ty

Bài tập 2: Mô tả yêu cầu

Xây dựng interface tên **Actionable**, bao gồm

- **Phương thức calculateTax:** Tính thuế của 1 loại phương tiện
- **Phương thức calculatePrice:** Tính tổng tiền của 1 loại phương tiện
- **Phương thức getInfor:** Trả về thông tin của phương tiện

Xây dựng lớp **Car** hiện thực từ **Actionable**, bao gồm:

- **Properties:** Name – Tên, Producer – Hãng, Year – Năm SX, Seat – Số chỗ ngồi, Price – Giá.
- **Hiện thực phương thức calculateTax** như quy tắc sau:
 - Nếu số chỗ ngồi dưới 7 chỗ thì thuế = Giá gốc * 60%.
 - Ngược lại thì thuế = Giá gốc * 70%.
- **Hiện thực phương thức calculatePrice** để tính giá tiền theo công thức:
Tổng tiền = giá gốc + thuế.
- **Hiện thực phương thức getInfor** để hiển thị thông tin theo mẫu: "... car produced by ... in ... has ... seats with the total price is ... \$".

Ví dụ: *Ford car produced by Ford in 1997 has 4 seats with the total price is 20000\$.*

Lớp **CarManagement** chứa phương thức main()

- **Tạo danh sách Car.**
- **Hiển thị danh sách Car**
- **Tính thuế**
- **Tính giá**