Marco Martinez
CISP 401
11/29/18
Assignment 3

CLASS DIAGRAM

```
Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver

Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

EmployeeRecord Class Attributes
      INSTANCE VARIABLES
      (+) String lastName
      (+) String firstName
      (+) double grossPay
      (+) double taxAmt
      (+) double netPay
      (+) int    employeeNumber
      (+) char   type

      CLASS CONSTRUCTORS
      (+) EmployeeRecord()
      (+) EmployeeRecord(String newLastName, String newFirstName, double newGrossPay, char newType)
      (+) EmployeeRecord(EmployeeRecord e)

      READ STATE SERVICES
      (+) String toString()

Employee Class Attributes
      CONSTANT DEFINITIONS
      (-) double TAXRATE

      INSTANCE VARIABLES
      (#) EmployeeRecord e

      CHANGE STATE SERVICES
      (+) abstract void calcGross()
      (+) void calcTaxes()
      (+) void calcNet()

      READ STATE SERVICES
      (+) EmployeeRecord get()
      (+) String toString()
```

Salary Class Attributes

CONSTANT DEFINITIONS
```
(-) char TYPE = 's';
```

INSTANCE VARIABLE DECLARATIONS
```
(-) double salary;
```

CLASS CONSTRUCTORS
```
(+) Piece()
(+) Piece(String lastName, String firstName, double newSalary)
(+) Piece(EmployeeRecord newEmployeeRecord)
(+) Piece(Employee newEmployee)
```

CHANGE STATE SERVICES
```
(+) void calcGross()
```

READ STATE SERVICES
```
(+) double getRate()
```

Piece Class Attributes

CONSTANT DEFINITIONS
```
(-) char TYPE
```

INSTANCE VARIABLE DECLARATION
```
(-) double pricePerPiece;
(-) int    pieces;
```

CLASS CONSTRUCTORS
```
(+) Piece()
(+) Piece(String newLastName, String newFirstName, double newPieceRate, int newNumPieces)
(+) Piece(EmployeeRecord newEmployeeRecord)
(+) Piece(Employee newEmployee)
```

CHANGE STATE SERVICES
```
(+) void calcGross()
```

READ STATE SERVICES
```
(+) double getPrice()
(+) int getPieces()
```

Hourly Class Attributes

CONSTANT DEFINITIONS
```
(-) double REGULARHOURS
(-) double OVERTIMERATE
(-) char TYPE
```

INSTANCE VARIABLE DECLARATIONS
```
(-) double hours;
(-) double rate;
```

CLASS CONSTRUCTORS
```
(+) Employee()
(+) Employee(String lastName, String firstName, double hrsWkd, double payRate)
(+) Employee(EmployeeRecord newEmployeeRecord)
(+) Employee(Employee newEmployee)
```

CHANGE STATE SERVICES
```
(+) void calcGross()
```

READ STATE SERVICES
```
(+) double getRate()
(+) double getHours()
```

Employees Class Attributes

```
      INSTANCE VARIABLES
      (-) EmployeeRecord[] emps
      (-) int index
      (-) int currentIndex

      CLASS CONSTRUCTORS
      (+) Employees()
      (+) Employees(Employee newEmployee)
      (+) Employees(Employee[] newEmployee, int newIndex)
      (+) Employees(Employees newEmployees)

      CHANGE STATE SERVICES
      (+) void add(Employee e)
      (+) void add(Employee[] newEmployee, int newIndex)
      (+) void add(Employees newEmployees)
      (+) void delete(String lastName)
      (+) void delete(int eID)
      (+) void sort()
      (-) void qSort(int start, int finish)

      READ STATE SERVICES
      (+) int getLength()
      (+) Employee[] getEmp()
      (+) Employee getEmp(int indexPoint)
      (+) EmployeeRecord search(int eID)
      (+) EmployeeRecord search(String lastName)
      (-) EmployeeRecord biSearch(int key, int low, int high)
      (-) EmployeeRecord biSearch(String key, int low, int high)
      (+) int getCurrentIndex()
      (+) EmployeeRecord iterate()
      (+) boolean hasNext()
      (+) void resetIterator()
      (+) String toString()
```

STATE MODEL

EmployeeRecord

EmployeeRecord() → s(null)
EmployeeRecord(String, String, double, double) → s0
EmployeeRecord(EmployeeRecord) → s0
s0 → toString() → s(terminal)
Employee
s3 → calcGross() → s3
s3 → calcTax() → s3
s3 → calcNet() → s3
s3 → get() → s(terminal)
s3 → toString() → s(terminal)

Hourly

Hourly() → s(null)
Hourly(String, String, double, double) → s3 // Processes are applied upon creation Hourly(EmployeeRecord) → s3 // Processes are applied upon creation (if applicable)
Hourly(Employee) → s3 // Processes are applied upon creation (if applicable)
s3 → calcGross() → s3
s3 → calcTax() → s3
s3 → calcNet() → s3
s3 → getRate() → s(terminal)
s3 → getHours Salary Salary() → s(null)

Salary
Salary(String, String, double, double) → s3 // Processes are applied upon creation Salary(EmployeeRecord) → s3 // Processes are applied upon creation (if applicable)
Salary(Employee) → s3 // Processes are applied upon creation (if applicable)
s3 → calcGross() → s3
s3 → getPiece() → s(terminal)

Employees
Employees() → s(null)
Employees(Employee) → s0
Employees(Employee[], int) → s0
Employees(Employees) → s0
s0 → add(Employee) → s0
s0 → add(Employee[],int) → s0
s0 → add(Employees) → s0
s0 → delete(String) → s(err)
s0 → delete(int) → s(err)
s0 → sort() → s1
s0 → getLength() → s(terminal)
s0 → getEmp() → s(terminal)
s0 → search(int) → s(err)
s0 → search(String) → s(err)
s0 → iterate() → s(terminal)
s0 → hasNext() → s(terminal)
s0 → resetIterator() → s(terminal)
s0 → toString() → s(terminal)
s1 → add(Employee) → s0
s1 → add(Employee[],int) → s0
s1 → add(Employees) → s0
s1 → delete(String) → s0
s1 → delete(int) → s0
s1 → sort() → s1
s1 → getLength() → s(terminal)
s1 → getEmp() → s(terminal)
s1 → search(int) → s(terminal)
s1 → search(String) → s(terminal)
s1 → iterate() → s(terminal)
s1 → hasNext() → s(terminal)
s1 → resetIterator() → s(terminal)
s1 → toString() → s(terminal)

Use Case Model (Hourly)

Normal Scenario 1:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction.
3. User requests the processed values via get() method.
4. User exits application.

Normal Scenario 2:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests the processed values via get() method.
4. User exits application.

Normal Scenario 3:

1. User inputs Employee object.
2. User requests the processed values via get() method.
3. User exits application.

Normal Scenario 4:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant grossPay calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 5:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant grossPay calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 6:

1. User inputs Employee object.
2. User requests redundant grossPay calculation.
3. User requests the processed values via get() method.
4. User exits application.

Normal Scenario 7:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant taxAmt calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 8:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant taxAmt calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 9:

1. User inputs Employee object.
2. User requests redundant taxAmt calculation.
3. User requests the processed values via get() method.
4. User exits application.

Normal Scenario 10:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant netPay calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 11:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant netPay calculation
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 12:

1. User inputs Employee object.
2. User requests redundant netPay calculation
3. User requests the processed values via get() method.
4. User exits application.

Normal Scenario 13:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant grossPay calculation.
4. User requests redundant taxAmt calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 14:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant grossPay calculation.
4. User requests redundant taxAmt calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 15:

1. User inputs Employee object.
2. User requests redundant grossPay calculation.
3. User requests redundant taxAmt calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 16:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant taxAmt calculation.
4. User requests redundant grossPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 17:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant taxAmt calculation.
4. User requests redundant grossPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 18:

1. User inputs Employee object.
2. User requests redundant taxAmt calculation.
3. User requests redundant grossPay calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 19:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant grossPay calculation.
4. User requests redundant netPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 20:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant grossPay calculation.
4. User requests redundant netPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 21:

1. User inputs Employee object.
2. User requests redundant grossPay calculation.
3. User requests redundant netPay calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 22:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant taxAmt calculation.
4. User requests redundant netPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 23:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant taxAmt calculation.
4. User requests redundant netPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 24:

1. User inputs Employee object.
2. User requests redundant taxAmt calculation.
3. User requests redundant netPay calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 25:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant netPay calculation.
4. User requests redundant taxAmt calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 26:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant netPay calculation.
4. User requests redundant taxAmt calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 27:

1. User inputs Employee object.
2. User requests redundant netPay calculation.
3. User requests redundant taxAmt calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 28:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant netPay calculation.
4. User requests redundant grossPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 29:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant netPay calculation.
4. User requests redundant grossPay calculation.
5. User requests the processed values via get() method.
6. User exits application.

Normal Scenario 30:

1. User inputs Employee object.
2. User requests redundant netPay calculation.
3. User requests redundant grossPay calculation.
4. User requests the processed values via get() method.
5. User exits application.

Normal Scenario 31:

1. User inputs 2 String values and 2 double values.
2. Processes are applied upon construction of object.
3. User requests redundant grossPay calculation.
4. User requests redundant taxAmt calculation.
5. User requests redundant netPay calculation.
6. User requests the processed values via get() method.
7. User exits application.

Normal Scenario 32:

1. User inputs EmployeeRecord object.
2. Processes are applied upon construction depending upon value of grossPay.
3. User requests redundant grossPay calculation.
4. User requests redundant taxAmt calculation.
5. User requests redundant netPay calculation
6. User requests the processed values via get() method.
7. User exits application.

Normal Scenario 33:

1. User inputs Employee object.
2. User requests redundant grossPay calculation.
3. User requests redundant taxAmt calculation.
4. User requests redundant netPay calculation
5. User requests the processed values via get() method.
6. User exits application.

Abnormal Scenario 1:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User requests processed information via get() method and is returned null values.
5. User exits program.

Abnormal Scenario 2:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User requests processed information via get() method and is returned null values.
5. User exits program.

Abnormal Scenario 3:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User requests processed information via get() method and is returned null values.
5. User exits program.

Abnormal Scenario 4:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newPayRate as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User requests processed information via get() method and is returned null values.
5. User exits program.

Abnormal Scenario 5:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User requests processed information via get() method and is returned null values.
6. User exits program.

Abnormal Scenario 6:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User requests processed information via get() method and is returned null values.
6. User exits program.

Abnormal Scenario 7:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User requests processed information via get() method and is returned null values.
6. User exits program.

Abnormal Scenario 8:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User requests processed information via get() method and is returned null values.
6. User exits program.

Abnormal Scenario 9:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User requests processed information via get() method and is returned null values.
6. User exits program.

Abnormal Scenario 10:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User requests processed information via get() method and is returned null values.
6. User exits program.

Abnormal Scenario 11:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User requests processed information via get() method and is returned null values.
7. User exits program.

Abnormal Scenario 12:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User requests processed information via get() method and is returned null values.
7. User exits program.

Abnormal Scenario 13:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User requests processed information via get() method and is returned null values.
7. User exits program.

Abnormal Scenario 14:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User requests processed information via get() method and is returned null values.
7. User exits program.

Abnormal Scenario 15:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor recognizes newPayRate as invalid (less than 0).
6. Constructor assigns an uninitialized EmployeeRecord to e.
7. User requests processed information via get() method and is returned null values.
8. User exits program.

Abnormal Scenario 16:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate grossPay and receives an error.
5. User exits program.

Abnormal Scenario 17:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate grossPay and receives an error.
5. User exits program.

Abnormal Scenario 18:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate grossPay and receives an error.
5. User exits program.

Abnormal Scenario 19:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newPayRate as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate grossPay and receives an error.
5. User exits program.

Abnormal Scenario 20:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate grossPay and receives an error.
6. User exits program.

Abnormal Scenario 21:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate grossPay and receives an error.
6. User exits program.

Abnormal Scenario 22:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate grossPay and receives an error.
6. User exits program.

Abnormal Scenario 23:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate grossPay and receives an error.
6. User exits program.

Abnormal Scenario 24:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate grossPay and receives an error.
6. User exits program.

Abnormal Scenario 25:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate grossPay and receives an error.
6. User exits program.

Abnormal Scenario 26:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate grossPay and receives an error.
7. User exits program.

Abnormal Scenario 27:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate grossPay and receives an error.
7. User exits program.

Abnormal Scenario 28:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate grossPay and receives an error.
7. User exits program.

Abnormal Scenario 29:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate grossPay and receives an error.
7. User exits program.

Abnormal Scenario 30:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor recognizes newPayRate as invalid (less than 0).
6. Constructor assigns an uninitialized EmployeeRecord to e.

7 User attempts to calculate grossPay and receives an error.

8. User exits program.

Abnormal Scenario 31:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate taxAmt and receives an error.
5. User exits program.

Abnormal Scenario 32:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate taxAmt and receives an error.
5. User exits program.

Abnormal Scenario 33:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate taxAmt and receives an error.
5. User exits program.

Abnormal Scenario 34:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newPayRate as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate taxAmt and receives an error.
5. User exits program.

Abnormal Scenario 35:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate taxAmt and receives an error.
6. User exits program.

Abnormal Scenario 36:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate taxAmt and receives an error.
6. User exits program.

Abnormal Scenario 37:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate taxAmt and receives an error.
6. User exits program.

Abnormal Scenario 38:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate taxAmt and receives an error.
6. User exits program.

Abnormal Scenario 39:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate taxAmt and receives an error.
6. User exits program.

Abnormal Scenario 40:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate taxAmt and receives an error.
6. User exits program.

Abnormal Scenario 41:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate taxAmt and receives an error.
7. User exits program.

Abnormal Scenario 42:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate taxAmt and receives an error.
7. User exits program.

Abnormal Scenario 43:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate taxAmt and receives an error.
7. User exits program.

Abnormal Scenario 44:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate taxAmt and receives an error.
7. User exits program.

Abnormal Scenario 45:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor recognizes newPayRate as invalid (less than 0).
6. Constructor assigns an uninitialized EmployeeRecord to e.

7 User attempts to calculate taxAmt and receives an error.

8. User exits program.

Abnormal Scenario 46:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate netPay and receives an error.
5. User exits program.

Abnormal Scenario 47:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate netPay and receives an error.
5. User exits program.

Abnormal Scenario 48:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate netPay and receives an error.
5. User exits program.

Abnormal Scenario 49:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newPayRate as invalid (less than 0).
3. Constructor assigns an uninitialized EmployeeRecord to e.
4. User attempts to calculate netPay and receives an error.
5. User exits program.

Abnormal Scenario 50:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate netPay and receives an error.
6. User exits program.

Abnormal Scenario 51:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate netPay and receives an error.
6. User exits program.

Abnormal Scenario 52:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate netPay and receives an error.
6. User exits program.

Abnormal Scenario 53:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate netPay and receives an error.
6. User exits program.

Abnormal Scenario 54:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate netPay and receives an error.
6. User exits program.

Abnormal Scenario 55:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newHrsWkd as invalid (less than 0).
3. Constructor recognizes newPayRate as invalid (less than 0).
4. Constructor assigns an uninitialized EmployeeRecord to e.
5. User attempts to calculate netPay and receives an error.
6. User exits program.

Abnormal Scenario 56:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate netPay and receives an error.
7. User exits program.

Abnormal Scenario 57:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate netPay and receives an error.
7. User exits program.

Abnormal Scenario 58:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate netPay and receives an error.
7. User exits program.

Abnormal Scenario 59:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newFirstName as invalid (not alphanumerical).
3. Constructor recognizes newHrsWkd as invalid (less than 0).
4. Constructor recognizes newPayRate as invalid (less than 0).
5. Constructor assigns an uninitialized EmployeeRecord to e.
6. User attempts to calculate netPay and receives an error.
7. User exits program.

Abnormal Scenario 60:

1. User inputs 2 String values and 2 double values as newLastName, newFirstName, newHrsWkd, newPayRate.
2. Constructor recognizes newLastName as invalid (not alphanumerical).
3. Constructor recognizes newFirstName as invalid (not alphanumerical).
4. Constructor recognizes newHrsWkd as invalid (less than 0).
5. Constructor recognizes newPayRate as invalid (less than 0).
6. Constructor assigns an uninitialized EmployeeRecord to e.

7 User attempts to calculate netPay and receives an error.

8. User exits program.

Java Source Code

```
/**
   @author       Marco Martinez
   @fileName     EmployeeRecord.java
   @version      1.1
   @description  This program will construct and manipulate EmployeeRecord objects.

   Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver

   Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

   EmployeeRecord Class Attributes
      INSTANCE VARIABLES
      (+) String lastName
      (+) String firstName
      (+) double grossPay
      (+) double taxAmt
      (+) double netPay
      (+) int    employeeNumber
      (+) char   type

      CLASS CONSTRUCTORS
      (+) EmployeeRecord()
      (+) EmployeeRecord(String newLastName, String newFirstName, double newGrossPay, char newType)
      (+) EmployeeRecord(EmployeeRecord e)

      READ STATE SERVICES
      (+) String toString()

   @date         10/11/2018

   Program Change Log
   ==========================
   Name     Date     Description
   Marco    10/11    Create baseline for EmployeeRecord.
   Marco    11/12    Adjust for inheritance.
 */

public class EmployeeRecord
{
   // INSTANCE VARIABLE DECLARATIONS
   public String lastName,
                 firstName;
   public double grossPay,
                 taxAmt,
                 netPay;
   public int    employeeNumber;
   public char   type;

   // CLASS CONSTRUCTORS
   // (+) EmployeeRecord()
   public EmployeeRecord(){}

   // (+) EmployeeRecord(String newLastName, String newFirstName, char newType)
   public EmployeeRecord(String newLastName, String newFirstName, char newType)
   {
      if ((Character.toLowerCase(newType) != 'h' && Character.toLowerCase(newType) != 'p' &&
Character.toLowerCase(newType) != 's') || !newLastName.matches("[a-zA-Z]+") || !newFirstName.matches("[a-zA-
Z]+")) return;
      else
      {
         this.lastName = newLastName;
         this.firstName = newFirstName;
         this.type = newType;
         this.grossPay = this.taxAmt = this.netPay = 0.00;
      }
   }

   // (+) EmployeeRecord(EmployeeRecord newEmployeeRecord)
   public EmployeeRecord(EmployeeRecord newEmployeeRecord)
   {
      this.lastName = newEmployeeRecord.lastName;
      this.firstName = newEmployeeRecord.firstName;
      this.grossPay = newEmployeeRecord.grossPay;
      this.taxAmt = newEmployeeRecord.taxAmt;
      this.netPay = newEmployeeRecord.netPay;
      this.employeeNumber = newEmployeeRecord.employeeNumber;
      this.type = newEmployeeRecord.type;
   }

   // READ STATE SERVICES
   // (+) String toString()
   public String toString()
   {
      return this.lastName + ", " + this.firstName
                           + " " + Double.toString(this.grossPay)
                           + " " + Double.toString(this.taxAmt)
                           + " " + Double.toString(this.netPay);
   }
}
```

```
/**
   @author       Marco Martinez
   @fileName     Employee.java
   @version      1.0
   @description  This program will construct and manipulate Employee objects.

   Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver

   Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

   Employee Class Attributes
      CONSTANT DEFINITIONS
      (-) double TAXRATE

      INSTANCE VARIABLES
      (#) EmployeeRecord e

      CHANGE STATE SERVICES
      (+) abstract void calcGross()
      (+) void calcTaxes()
      (+) void calcNet()

      READ STATE SERVICES
      (+) EmployeeRecord get()
      (+) String toString()

   @date         10/11/2018

   Program Change Log
   ==========================
   Name     Date     Description
   Marco    10/11    Create baseline for Employee.
   Marco    11/12    Adjust for inheritance.
 */

public abstract class Employee
{
   // CONSTANT DEFINITIONS
   private static final double TAXRATE = 0.15;

   // INSTANCE VARIABLE DECLARATIONS
   protected EmployeeRecord e;

   // CLASS CONSTRUCTORS
   // (+) Employee()
   public Employee(){}

   // (+) Employee(String newLastName, String newFirstName, char newType)
   public Employee(String newLastName, String newFirstName, char newType)
   {
      if ((Character.toLowerCase(newType) != 'h' && Character.toLowerCase(newType) != 'p' &&
Character.toLowerCase(newType) != 's') || !newLastName.matches("[a-zA-Z]+") || !newFirstName.matches("[a-zA-
Z]+")) this.e = new EmployeeRecord();
      else
      {
         this.e = new EmployeeRecord(newLastName, newFirstName, newType);
      }
   }

   // (+) Employee(EmployeeRecord newEmployeeRecord)
   public Employee(EmployeeRecord newEmployeeRecord)
   {
      this.e = new EmployeeRecord(newEmployeeRecord);
      if (this.e.grossPay == 0) calcGross();
      if (this.e.taxAmt == 0) calcTax();
      if (this.e.netPay == 0) calcNet();
   }

   // (+) Employee(Employee newEmployee)
   public Employee(Employee newEmployee)
   {
      this.e = new EmployeeRecord(newEmployee.get());
      if (this.e.grossPay == 0) calcGross();
      if (this.e.taxAmt == 0) calcTax();
      if (this.e.netPay == 0) calcNet();
   }

   // CHANGE STATE SERVICES
   // (+) abstract void calcGross()
   public abstract void calcGross();

   // (+) void calcTax()
   public void calcTax()
   {
      this.e.taxAmt = this.e.grossPay * TAXRATE;
   }

   // (+) void calcNet()
   public void calcNet()
   {
      this.e.netPay = this.e.grossPay - this.e.taxAmt;
   }

   // READ STATE SERVICES

   // (+) EmployeeRecord get()
   public EmployeeRecord get()
   {
      return this.e;
   }

   // (+) String toString()
   public String toString()
   {
      return this.e.toString();
   }
}

/**
   @author       Marco Martinez
   @fileName     Salary.java
   @version      1.0
   @description  This program will construct and manipulate Salary-Employee objects.

   Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver
   Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

   Salary Class Attributes
      CONSTANT DEFINITIONS
      (-) char TYPE = 's';

      INSTANCE VARIABLE DECLARATIONS
      (-) double salary;

      CLASS CONSTRUCTORS
      (+) Piece()
      (+) Piece(String lastName, String firstName, double newSalary)
      (+) Piece(EmployeeRecord newEmployeeRecord)
      (+) Piece(Employee newEmployee)

      CHANGE STATE SERVICES
      (+) void calcGross()

      READ STATE SERVICES
      (+) double getRate()

   @date         11/12/2018

   Program Change Log
   ==========================
   Name     Date     Description
   Marco    11/12    Create baseline for Piece.
 */

public class Salary extends Employee
{
   // CONSTANT DEFINITIONS
   private static final char TYPE = 's';

   // INSTANCE VARIABLE DECLARATIONS
   private double salary;

   // CLASS CONSTRUCTORS
   // (+) Salary()
   public Salary(){}

   // (+) Salary(String newLastName, String newFirstName, double newSalary)
   public Salary(String newLastName, String newFirstName, double newSalary)
   {
      super(newLastName, newFirstName, TYPE);
      if (newSalary < 0)
      {
         this.e = new EmployeeRecord();
         return;
      }
      else
      {
         this.salary = newSalary;
         calcGross();
         calcTax();
         calcNet();
      }
   }

   // (+) Salary(EmployeeRecord newEmployee)
   public Salary(EmployeeRecord newEmployee)
   {
      super(newEmployee);
   }

   // (+) Salary(Employee newEmployee)
   public Salary(Employee newEmployee)
   {
      super(newEmployee);
      this.salary = ((Salary)newEmployee).getSalary();
   }

   // CHANGE STATE SERVICES
   // (+) void calcGross()
   public void calcGross()
   {
      this.e.grossPay = this.salary;
   }

   // READ STATE SERVICES
   // (+) double getSalary()
   public double getSalary()
   {
      return this.salary;
   }
}

/**
   @author       Marco Martinez
   @fileName     Piece.java
   @version      1.0
   @description  This program will construct and manipulate Piece-Employee objects.

   Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver

   Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

   Piece Class Attributes
      CONSTANT DEFINITIONS
      (-) char TYPE

      INSTANCE VARIABLE DECLARATION
      (-) double pricePerPiece;
      (-) int    pieces;

      CLASS CONSTRUCTORS
      (+) Piece()
      (+) Piece(String newLastName, String newFirstName, double newPieceRate, int newNumPieces)
      (+) Piece(EmployeeRecord newEmployeeRecord)
      (+) Piece(Employee newEmployee)

      CHANGE STATE SERVICES
      (+) void calcGross()

      READ STATE SERVICES
      (+) double getPrice()
      (+) int getPieces()
   @date         11/12/2018

   Program Change Log
   ==========================
   Name     Date     Description
   Marco    11/12    Create baseline for Piece.
 */

public class Piece extends Employee
{
   // CONSTANT DEFINITIONS
   private static final char TYPE = 'p';

   // INSTANCE VARIABLE DECLARATION
   private double pricePerPiece;
   private int    pieces;

   // CLASS CONSTRUCTORS
   // (+) Piece()
   public Piece(){}

   // (+) Piece(String newLastName, String newFirstName, double newPieceRate, int newNumPieces)
   public Piece(String newLastName, String newFirstName, double newPieceRate, int newNumPieces)
   {
      super(newLastName, newFirstName, TYPE);
      if (newPieceRate < 0 || newNumPieces < 0)
      {
         this.e = new EmployeeRecord();
         return;
      }
      else
      {
         this.pricePerPiece = newPieceRate;
         this.pieces = newNumPieces;
         calcGross();
         calcTax();
         calcNet();
      }
   }

   // (+) Piece(EmployeeRecord newEmployee)
   public Piece(EmployeeRecord newEmployee)
   {
      super(newEmployee);
   }

   // (+) Piece(Employee newEmployee)
   public Piece(Employee newEmployee)
   {
      super(newEmployee);
      this.pricePerPiece = ((Piece)newEmployee).getPrice();
      this.pieces = ((Piece)newEmployee).getPieces();
   }

   // CHANGE STATE SERVICES
   // (+) void calcGross()
   public void calcGross()
   {
      this.e.grossPay = this.pricePerPiece * this.pieces;
   }

   // READ STATE SERVICES
   // (+) double getPrice()
   public double getPrice()
   {
      return this.pricePerPiece;
   }
   // (+) int getPieces()
   public int getPieces()
   {
      return pieces;
   }
}

/**
   @author       Marco Martinez
   @fileName     Hourly.java
   @version      1.0
   @description  This program will construct and manipulate Hourly-Employee objects.

   Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver

   Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

   Hourly Class Attributes
      CONSTANT DEFINITIONS
      (-) double REGULARHOURS
      (-) double OVERTIMERATE
      (-) char TYPE

      INSTANCE VARIABLE DECLARATIONS
      (-) double hours;
      (-) double rate;

      CLASS CONSTRUCTORS
      (+) Employee()
      (+) Employee(String lastName, String firstName, double hrsWkd, double payRate)
      (+) Employee(EmployeeRecord newEmployeeRecord)
      (+) Employee(Employee newEmployee)

      CHANGE STATE SERVICES
      (+) void calcGross()

      READ STATE SERVICES
      (+) double getRate()
      (+) double getHours()

   @date         11/12/2018

   Program Change Log
   ==========================
   Name     Date     Description
   Marco    11/12    Create baseline for Hourly.
 */

public class Hourly extends Employee
{
   // CONSTANT DEFINITIONS
   private static final double REGULARHOURS = 40.0;
   private static final double OVERTIMERATE = 1.5;
   private static final char TYPE = 'h';
```

```
// INSTANCE VARIABLE DECLARATIONS
private double hours;
private double rate;

// CLASS CONSTRUCTORS
// (+) Hourly()
public Hourly(){}

// (+) Hourly(String newLastName, String newFirstName, double newPayRate, double newHrsWkd)
public Hourly(String newLastName, String newFirstName, double newPayRate, double newHrsWkd)
{
   super(newLastName, newFirstName, TYPE);
   if (newPayRate < 0 || newHrsWkd < 0)
   {
      this.e = new EmployeeRecord();
      return;
   }
   else
   {
      this.rate = newPayRate;
      this.hours = newHrsWkd;
      calcGross();
      calcTax();
      calcNet();
   }
}

// (+) Hourly(EmployeeRecord newEmployee)
public Hourly(EmployeeRecord newEmployee)
{
   super(newEmployee);
}

// (+) Hourly(Employee newEmployee)
public Hourly(Employee newEmployee)
{
   super(newEmployee);
   this.rate = ((Hourly)newEmployee).getRate();
   this.hours = ((Hourly)newEmployee).getHours();
}

// CHANGE STATE SERVICES
// (+) void calcGross()
public void calcGross()
{
   if (this.hours <= REGULARHOURS)
   {
      this.e.grossPay = this.rate * this.hours;
   }
   else
   {
      this.e.grossPay = REGULARHOURS * this.rate;
      this.e.grossPay += (this.hours - REGULARHOURS) * this.rate * OVERTIMERATE;
   }
}

// READ STATE SERVICES
// (+) double getRate()
public double getRate()
{
   return this.rate;
}

// (+) double getHours()
public double getHours()
{
   return this.hours;
   }
}

/**
   @author       Marco Martinez
   @fileName     Employees.java
   @version      1.0
   @description  This program will construct and manipulate Employees objects.

   Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver

   Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

   Employees Class Attributes
      INSTANCE VARIABLES
      (-) EmployeeRecord[] emps
      (-) int index
      (-) int currentIndex

      CLASS CONSTRUCTORS
      (+) Employees()
      (+) Employees(Employee newEmployee)
      (+) Employees(Employee[] newEmployee, int newIndex)
      (+) Employees(Employees newEmployees)

      CHANGE STATE SERVICES
      (+) void add(Employee e)
      (+) void add(Employee[] newEmployee, int newIndex)
      (+) void add(Employees newEmployees)
      (+) void delete(String lastName)
      (+) void delete(int eID)
      (+) void sort()
      (-) void qSort(int start, int finish)

      READ STATE SERVICES
      (+) int getLength()
      (+) Employee[] getEmp()
      (+) Employee getEmp(int indexPoint)
      (+) EmployeeRecord search(int eID)
      (+) EmployeeRecord search(String lastName)
      (-) EmployeeRecord biSearch(int key, int low, int high)
      (-) EmployeeRecord biSearch(String key, int low, int high)
      (+) int getCurrentIndex()
      (+) EmployeeRecord iterate()
      (+) boolean hasNext()
      (+) void resetIterator()
      (+) String toString()

   @date         10/11/2018

   Program Change Log
   ==========================
   Name     Date     Description
   Marco    10/11    Create baseline for Employees.
   Marco    10/28    Add finishing touches to Employees.
   Marco    11/12    Adjust for inheritance model.
   Marco    12/8     Adjust per feedback. Removed multiple add and constructor methods. Removed
"determineEmployee" method.
 */

public class Employees
{
   // CONSTANT DEFINITIONS
   public static final int EMPMAX = 100;

   // INSTANCE VARIABLE INITIALIZATIONS
   private Employee emps[] = new Employee[EMPMAX];
   private int index = 0;
   private int currentIndex = 0;

   // CLASS CONSTRUCTORS
   // (+) Employees()
   public Employees(){}

   // (+) Employees(Employee e)
   public Employees(Employee e)
   {
      this.emps[this.index] = e;
      (this.emps[this.index].get()).employeeNumber = this.index;
      this.index++;
   }

   // (+) Employees(Employee[] newEmployee, int newIndex)
   public Employees(Employee[] newEmployee, int newIndex)
   {
      if (newIndex < EMPMAX)
      {
         for (int i = 0; i < newIndex; i++)
         {
            this.emps[this.index] = newEmployee[i];
            (this.emps[this.index].get()).employeeNumber = this.index;
            this.index++;
         }
      }
   }

   // (+) Employees(Employees newEmployees)
   public Employees(Employees newEmployees)
   {
      if (newEmployees.getLength() < EMPMAX)
      {
         for(int i = 0; i < newEmployees.getLength(); i++)
         {
            this.emps[this.index] = newEmployees.getEmp(i);
            (this.emps[this.index].get()).employeeNumber = this.index;
            this.index++;
         }
      }

   }

   // CHANGE STATE SERVICES
   // (+) void add(Employee e)
   public void add(Employee e)
   {
      this.emps[this.index] = e;
      (this.emps[this.index].get()).employeeNumber = this.index;
      this.index++;
   }

   // (+) void add(Employee[] newEmployee, int newIndex)
   public void add(Employee[] newEmployee, int newIndex)
   {
```

```
   if (this.index + newIndex < EMPMAX)
   {
      for (int i = 0; i < newIndex; i++)
      {
         this.emps[this.index] = newEmployee[i];
         (this.emps[this.index].get()).employeeNumber = this.index;
         this.index++;
      }
   }
}

// (+) void add(Employees newEmployees)
public void add(Employees newEmployees)
{
   if (this.index + newEmployees.getLength() < EMPMAX)
   {
      for(int i =  0; i < newEmployees.getLength(); i++)
      {
         this.emps[this.index] = newEmployees.getEmp(i);
         (this.emps[this.index].get()).employeeNumber = this.index;
         this.index++;
      }
   }
}

// (+) void delete(int eID)
public void delete(int eID)
{
   if ((search(eID).lastName) != null)
   {
      this.emps[search(eID).employeeNumber] = this.emps[index-1];
      this.emps[index-1] = new Hourly();
      this.index--;
   }
   else return;
}

// (+) void delete(String lastName)
public void delete(String lastName)
{
   if ((search(lastName).lastName) != null)
   {
      this.emps[(search(lastName)).employeeNumber] = this.emps[index-1];
      this.emps[index-1] = new Hourly();
      this.index--;
   }
   else return;
}

// (+) void sort()
public void sort()
{
   if (this.index > 0) qSort(0, this.index-1);
}

// (-)  void qSort(int start, int finish)
private void qSort(int start, int finish)
{
   int i = start;
   int j = finish;
   String pivot = (this.emps[start + (finish - start) / 2].get()).lastName;

   while (i <= j)
   {
      while ((this.emps[i].get()).lastName.compareToIgnoreCase(pivot) < 0)
      {
         i++;
      }
```

```
            while ((this.emps[j].get()).lastName.compareToIgnoreCase(pivot) > 0)
            {
               j--;
            }

            if (i <= j)
            {
               Employee temp = this.emps[i];
               this.emps[i] = this.emps[j];
               this.emps[j] = temp;
               i++;
               j--;
            }
         }

           if (start < j) {
               qSort(start, j);
           }
           if (i < finish) {
               qSort(i, finish);
           }

           for(int n = 0; n < this.index; n++) (this.emps[n].get()).employeeNumber = n;
      }

      // READ STATE SERVICES
      // (+) int getLength()
      public int getLength()
      {
         return this.index;
      }

      // (+) Employee[] getEmp()
      public Employee[] getEmp()
      {
         return this.emps;
      }

      // (+) Employee getEmp(int indexPoint)
      public Employee getEmp(int indexPoint)
      {
         return this.emps[indexPoint];
      }

      // (+) EmployeeRecord search(int eID)
      public EmployeeRecord search(int eID)
      {
         return biSearch(eID, 0, this.index-1);
      }

      // (+) EmployeeRecord search(String lastName)
      public EmployeeRecord search(String lastName)
      {
         return biSearch(lastName, 0, this.index-1);
      }

      // (-)  EmployeeRecord biSearch(int key, int low, int high)
      private EmployeeRecord biSearch(int key, int low, int high)
      {
         while(high >= low)
         {
            int middle = (low + high) / 2;
            if ((this.emps[middle].get()).employeeNumber == key)
            {
               return this.emps[middle].get();
            }
            if ((this.emps[middle].get()).employeeNumber > key)
            {
               return biSearch(key, low, middle-1);
            }
            if ((this.emps[middle].get()).employeeNumber < key)
            {
               return biSearch(key, middle+1, high);
            }
         }
         return new EmployeeRecord();
      }

      // (-)  EmployeeRecord biSearch(String key, int low, int high)
      private EmployeeRecord biSearch(String key, int low, int high)
      {
         while(high >= low)
         {
            int middle = (low + high) / 2;
            if ((this.emps[middle].get()).lastName.compareToIgnoreCase(key) == 0)
            {
               return this.emps[middle].get();
            }
            if ((this.emps[middle].get()).lastName.compareToIgnoreCase(key) > 0)
            {
               return biSearch(key, low, middle-1);
            }
            if ((this.emps[middle].get()).lastName.compareToIgnoreCase(key) < 0)
            {
               return biSearch(key, middle+1, high);
            }
         }
         return new EmployeeRecord();
      }

      // (+) int getCurrentIndex()
      public int getCurrentIndex()
      {
         return this.currentIndex;
      }

      // (+) EmployeeRecord iterate()
      public EmployeeRecord iterate()
      {
         EmployeeRecord temp = new EmployeeRecord(this.emps[this.currentIndex].get());
         this.currentIndex++;
         return temp;
      }

      // (+) void resetIterator()
      public void resetIterator()
      {
         this.currentIndex = 0;
      }

      // (+) boolean hasNext()
      public boolean hasNext()
      {
         if (this.currentIndex < this.index)
         {
            return true;
         }
         else
         {
            return false;
         }
      }

      // (+) String toString()
      public String toString()
```

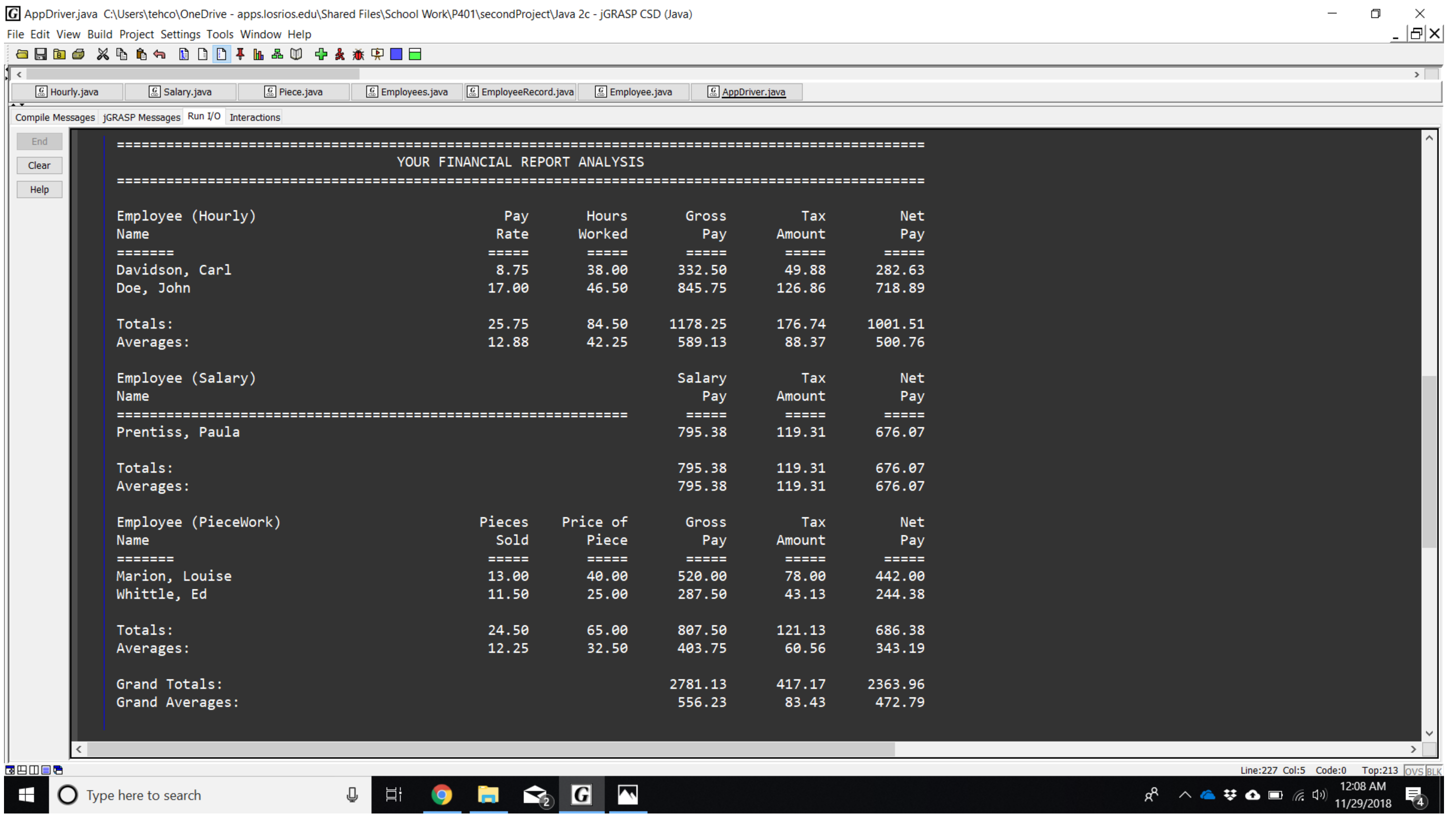
```
   {
      String str = "" ;

      for(int i = 0; i < this.index; i++) str += "\n" + this.emps[i].toString();

      return str;
   }
}

/**
   @author       Marco Martinez
   @fileName     AppDriver.java
   @version      1.0
   @description  This program will utilize Employees objects for creating a report.

   Classes
      EmployeeRecord
      Employee
      Hourly
      Salary
      Piece
      Employees
      AppDriver

   Associations
      EmployeeRecord(1) --- includes --- (1) Employee
      Hourly(1) --- inherits --- (1) Employee
      Salary(1) --- inherits --- (1) Employee
      Piece(1) --- inherits --- (1) Employee
      Employees(1) --- contains --- (m) Employee
      AppDriver(1) --- uses --- (1) Employees

   AppDriver Class
      (+) static void getEmployees(Employees myEmps)
      (+) static void getEmployee(Employees myEmps, String payPrompt, String numOfPrompt, char tempType,
Scanner input)
      (-) static Employee determineEmployee(String lastName, String firstName, double payRate, double hrsWkd,
char c)
      (+) static char validateAnswer(char c, Scanner input)
      (+) static String validateString(String name, Scanner input)
      (+) static double validateDouble(double value, Scanner input)
      (+) static char validateYesNo(char c, Scanner input)
      (+) static void printReport(Employees myEmps)
      (+) static void printHeading()
      (+) static void printLabels()
      (+) static void printEmployees(Employees myEmps)
      (+) static boolean determineIfTypeExists(Employees myEmps, char type)
      (+) static void printEmployee(EmployeeRecord emp, String str, char type)
      (+) static void printTypeFooter(Employees myEmps, char type)
      (+) static void printTotals(double totalRate, double totalQuantity, double totalGross, double totalTax,
double totalNet, char type)
      (+) static void printAverages(double totalRate, double totalQuantity, double totalGross, double
totalTax, double totalNet, int empNum, char type)
      (+) static void printFooter(Employees myEmps)
      (+) static void printString38(String str)
      (+) static void printString12(String str)
      (+) static void printDouble12(double value)
      (+) static String concatenateName(String lastName, String firstName)


   @date         10/11/2018

   Program Change Log
   ==========================
   Name     Date     Description
   Marco    10/11    Create baseline for AppDriver.
   Marco    10/28    Add finishing touches to AppDriver.
   Marco    11/13    Adjust for inheritance.
   Marco    12/8     Adjust for feedback. Moved "determineEmployee" method.
 */
// LIBRARIES
import java.util.Scanner; // Allows access to scanner

public class AppDriver
{
   // CONSTANT DEFINITIONS
   public static final String TYPE_PROMPT = new String("Please enter the employee's type here: ");
   public static final String FIRST_PROMPT = new String("Please enter the employee's first name here: ");
   public static final String LAST_PROMPT = new String("Please enter the employee's last name here: ");
   public static final String HOUR_RATE_PROMPT = new String("Please enter the employee's hourly pay here: ");
   public static final String HOUR_HRS_PROMPT = new String("Please enter the employee's number of worked
hours: ");
   public static final String PIECE_RATE_PROMPT = new String("Please enter the employee's pay per piece of
work here: ");
   public static final String PIECE_NUM_PROMPT = new String("Please enter the employee's number of pieces
here: ");
   public static final String SALARY_PROMPT = new String("Please enter the employee's salary here: ");
   public static final String HOURLY_LABEL_TOP = new String(String.format("%-38s","Employee (Hourly)") +
String.format("%12s","Pay") + String.format("%12s","Hours") + String.format("%12s","Gross") +
String.format("%12s","Tax") + String.format("%12s","Net"));
   public static final String HOURLY_LABEL_MIDDLE = new String(String.format("%-38s","Name") +
String.format("%12s","Rate") + String.format("%12s","Worked") + String.format("%12s","Pay") +
String.format("%12s","Amount") + String.format("%12s","Pay"));
   public static final String HOURLY_LABEL_BOTTOM = new String(String.format("%-38s","=======") +
String.format("%12s","=====") + String.format("%12s","=====") + String.format("%12s","=====") +
String.format("%12s","=====") + String.format("%12s","====="));
   public static final String SALARY_LABEL_TOP = new String(String.format("%-38s","Employee (Salary)") +
String.format("%12s","") + String.format("%12s","") + String.format("%12s","Salary") +
String.format("%12s","Tax") + String.format("%12s","Net"));
   public static final String SALARY_LABEL_MIDDLE = new String(String.format("%-38s","Name") +
String.format("%12s","") + String.format("%12s","") + String.format("%12s","Pay") +
String.format("%12s","Amount") + String.format("%12s","Pay"));
   public static final String SALARY_LABEL_BOTTOM = new String(String.format("%-
38s","=======================================") + String.format("%12s","============") +
String.format("%12s","============") + String.format("%12s","=====") + String.format("%12s","=====") +
String.format("%12s","====="));
   public static final String PIECE_LABEL_TOP = new String(String.format("%-38s","Employee (PieceWork)") +
String.format("%12s","Pieces") + String.format("%12s","Price of") + String.format("%12s","Gross") +
String.format("%12s","Tax") + String.format("%12s","Net"));
   public static final String PIECE_LABEL_MIDDLE = new String(String.format("%-38s","Name") +
String.format("%12s","Sold") + String.format("%12s","Piece") + String.format("%12s","Pay") +
String.format("%12s","Amount") + String.format("%12s","Pay"));
   public static final String PIECE_LABEL_BOTTOM = new String(String.format("%-38s","=======") +
String.format("%12s","=====") + String.format("%12s","=====") + String.format("%12s","=====") +
String.format("%12s","=====") + String.format("%12s","====="));

   public static void main(String[] args)
   {
      // VARIABLE DECLARATIONS
      Employees myEmps = new Employees();

      // CALLS
      getEmployees(myEmps);
      myEmps.sort();
      printReport(myEmps);
   }

   // METHODS
   // (+) static void getEmployees(Employees myEmps)
   public static void getEmployees(Employees myEmps)
   {
      Scanner input = new Scanner(System.in);
      char c = 'y';
      char tempType;

      while (c != 'n' && c != 'N')
      {
         System.out.print(TYPE_PROMPT);
         tempType = validateAnswer(Character.toLowerCase(input.next().charAt(0)), input);

         switch (Character.toLowerCase(tempType))
         {
            case 'h':
               getEmployee(myEmps, HOUR_RATE_PROMPT, HOUR_HRS_PROMPT, 'h', input);
               break;
            case 's':
               getEmployee(myEmps, SALARY_PROMPT, "", 's', input);
               break;
            case 'p':
               getEmployee(myEmps, PIECE_RATE_PROMPT, PIECE_NUM_PROMPT, 'p', input);
               break;
            default:
               System.out.println("Error found in getEmployees(Employees) switch statement.");
               break;
         }

         System.out.print("Would you like to continue? (Y/N) ");
         c = validateYesNo((input.next().charAt(0)), input);
         System.out.println();

         if (myEmps.getLength() >= myEmps.EMPMAX - 1)
         {
            System.out.println("You have hit the maximum amount of employees to enter.");
            c = 'n';
         }
      }
      input.close();
   }

   // (+) static void getEmployee(Employees myEmps, String payPrompt, String numOfPrompt, char c, Scanner
input)
   public static void getEmployee(Employees myEmps, String payPrompt, String numOfPrompt, char c, Scanner
input)
   {
      String tempFirstName = new String("");
      String tempLastName = new String("");
      Double tempHrsWkd = 0.00;
      Double tempPayRate = 0.00;

      System.out.print(FIRST_PROMPT);
      tempFirstName = validateString(input.next(), input);

      System.out.print(LAST_PROMPT);
      tempLastName = validateString(input.next(), input);

      System.out.print(payPrompt);
      tempPayRate = validateDouble(input.nextDouble(), input);

      if (Character.toLowerCase(c) != 's')
      {
         System.out.print(numOfPrompt);
         tempHrsWkd = validateDouble(input.nextDouble(), input);
      }

      myEmps.add(determineEmployee(tempLastName,tempFirstName,tempPayRate,tempHrsWkd,c));
   }

   // (-)  static Employee determineEmployee(String lastName, String firstName, double payRate, double
hrsWkd, char c)
   private static Employee determineEmployee(String lastName, String firstName, double payRate, double
hrsWkd, char c)
   {
      switch (c)
      {
         case 'h':
            return new Hourly(lastName, firstName, payRate, hrsWkd);
         case 's':
            return new Salary(lastName, firstName, payRate);
         case 'p':
            return new Piece(lastName, firstName, payRate, (int) hrsWkd);
         default:
            System.out.println("Error found within determineEmployee(String, String, double, double, char)
case.");
            break;
      }
      return new Hourly();
   }

   // (+) static char validateAnswer(char c, Scanner input)
   public static char validateAnswer(char c, Scanner input)
   {
      while (Character.toLowerCase(c) != 'h' && Character.toLowerCase(c) != 's' && Character.toLowerCase(c)
!= 'p')
      {
         System.out.println("Invalid employee type.");
         System.out.print("Please specify between hourly, piecework or salary: ");
         c = input.next().charAt(0);
      }
      return c;
   }

   // (+) static String validateString(String name, Scanner input)
   public static String validateString(String name, Scanner input)
   {
      for(int i = 0; i < 3; i++)
      {
         if (name.matches("[a-zA-Z]+")) return name;
         System.out.println("Error. A name must be alphanumeric.");
         System.out.print("Please enter a name with the correct specifications: ");
         name = input.next();
      }

      return "Default";
   }

   // (+) static double validateDouble(double value, Scanner input)
   public static double validateDouble(double value, Scanner input)
   {
      for(int i = 0; i < 3; i++)
      {
         if (value > 0.00) return value;
         System.out.println("Error. Value must be more than 0.");
         System.out.print("Please enter a value with the correct specifications: ");
         value = input.nextDouble();
      }

      return 0.00;
   }

   // (+) static char validateYesNo(char c, Scanner input)
   public static char validateYesNo(char c, Scanner input)
   {
      while (c != 'n' && c != 'N' && c != 'y' && c != 'Y')
      {
         System.out.println("Invalid input.");
         System.out.print("Please enter either a 'y' or 'n': ");
         c = input.next().charAt(0);
         System.out.println();
      }
      return c;
   }
   // (+) static void printReport(Employees myEmps)
   public static void printReport(Employees myEmps)
   {
      printHeading();
      printEmployees(myEmps);
      printFooter(myEmps);
   }

   // (+) static void printHeading()
   public static void printHeading()
   {

System.out.println("========================================================================================
=========");
      System.out.println("                                  YOUR FINANCIAL REPORT ANALYSIS");

System.out.println("========================================================================================
=========");
      System.out.println();
   }

   // (+) static void printEmployees(Employees myEmps)
   public static void printEmployees(Employees myEmps)
   {
   Employee temp;
   EmployeeRecord emp;
   int counter;
      for(int i = 0; i < 3; i++)
      {
         switch (i)
         {
            case 0:
               if (determineIfTypeExists(myEmps, 'h'))
               {
                  System.out.println(HOURLY_LABEL_TOP);
                  System.out.println(HOURLY_LABEL_MIDDLE);
                  System.out.println(HOURLY_LABEL_BOTTOM);
                  while(myEmps.hasNext())
                  {
                     emp = new EmployeeRecord(myEmps.iterate());
                     if (emp.type == 'h')
                     {
                        temp = new Hourly(myEmps.getEmp(myEmps.getCurrentIndex()-1));
                        printEmployee(emp,'h',((Hourly) temp).getRate(),((Hourly) temp).getHours());
                     }
                  }
                  myEmps.resetIterator();
                  System.out.println();
                  printTypeFooter(myEmps,'h');
                  System.out.println();
               }
               break;
            case 1:
               if (determineIfTypeExists(myEmps,'s'))
               {
                  System.out.println(SALARY_LABEL_TOP);
                  System.out.println(SALARY_LABEL_MIDDLE);
                  System.out.println(SALARY_LABEL_BOTTOM);
                  while(myEmps.hasNext())
                  {
                     emp = new EmployeeRecord(myEmps.iterate());
                     if (emp.type == 's')
                     {
                        temp = new Salary(myEmps.getEmp(myEmps.getCurrentIndex()-1));
                        printEmployee(emp,'s',((Salary) temp).getSalary(), 0.00);
                     }
                  }
                  myEmps.resetIterator();
```

```java
               System.out.println();
               printTypeFooter(myEmps, 's');
               System.out.println();
            }
            break;
         case 2:
            if (determineIfTypeExists(myEmps,'p'))
            {
               System.out.println(PIECE_LABEL_TOP);
               System.out.println(PIECE_LABEL_MIDDLE);
               System.out.println(PIECE_LABEL_BOTTOM);
               while(myEmps.hasNext())
               {
                  emp = new EmployeeRecord(myEmps.iterate());
                  if (emp.type == 'p')
                  {
                     temp = new Piece(myEmps.getEmp(myEmps.getCurrentIndex()-1));
                     printEmployee(emp,'p',((Piece) temp).getPrice(),((Piece) temp).getPieces());
                  }
               }
               myEmps.resetIterator();
               System.out.println();
               printTypeFooter(myEmps, 'p');
               System.out.println();
            }
            break;
         default:
            System.out.println("Error found within printEmployees(Employees) switch statement.");
            break;
      }
   }
}

// (+) static boolean determineIfTypeExists(Employees myEmps, char type)
public static boolean determineIfTypeExists(Employees myEmps, char type)
{
   EmployeeRecord emp;

   while(myEmps.hasNext())
   {
      emp = new EmployeeRecord(myEmps.iterate());
      if (emp.type == type)
      {
         myEmps.resetIterator();
         return true;
      }
   }
   myEmps.resetIterator();
   return false;
}

// (+) static void printEmployee(EmployeeRecord emp, char type, double rate, double quantity)
public static void printEmployee(EmployeeRecord emp, char type, double rate, double quantity)
{
   if (emp.type == type)
   {
      switch (type)
      {
         case 'h':
            printString38(concatenateName(emp.lastName, emp.firstName));
            printDouble12(rate);
            printDouble12(quantity);
            printDouble12(emp.grossPay);
            printDouble12(emp.taxAmt);
            printDouble12(emp.netPay);
            System.out.println();
            break;
         case 's':
            printString38(concatenateName(emp.lastName, emp.firstName));
            printString12("");
            printString12("");
            printDouble12(rate);
            printDouble12(emp.taxAmt);
            printDouble12(emp.netPay);
            System.out.println();
            break;
         case 'p':
            printString38(concatenateName(emp.lastName, emp.firstName));
            printDouble12(rate);
            printDouble12(quantity);
            printDouble12(emp.grossPay);
            printDouble12(emp.taxAmt);
            printDouble12(emp.netPay);
            System.out.println();
            break;
      }
   }
}

// (+) static void printTypeFooter(Employees myEmps, char type)
public static void printTypeFooter(Employees myEmps, char type)
{
   double totalRate = 0,
          totalQuantity = 0,
          totalGross = 0,
          totalTax = 0,
          totalNet = 0;
   int counter = 0;
   Employee temp;

   while(myEmps.hasNext())
   {
      if ((myEmps.iterate()).type == type)
      {
         counter++;
         switch (type)
         {
            case 'h':
               temp = new Hourly(myEmps.getEmp(myEmps.getCurrentIndex()-1));
               totalRate += ((Hourly)temp).getRate();
               totalQuantity += ((Hourly)temp).getHours();
               totalGross += (temp.get()).grossPay;
               totalTax += (temp.get()).taxAmt;
               totalNet += (temp.get()).netPay;
               break;
            case 's':
               temp = new Salary(myEmps.getEmp(myEmps.getCurrentIndex()-1));
               totalRate += ((Salary)temp).getSalary();
               totalGross += (temp.get()).grossPay;
               totalTax += (temp.get()).taxAmt;
               totalNet += (temp.get()).netPay;
               break;
            case 'p':
               temp = new Piece(myEmps.getEmp(myEmps.getCurrentIndex()-1));
               totalRate += ((Piece)temp).getPrice();
               totalQuantity += ((Piece)temp).getPieces();
               totalGross += (temp.get()).grossPay;
               totalTax += (temp.get()).taxAmt;
               totalNet += (temp.get()).netPay;
               break;
         }
      }
   }
   myEmps.resetIterator();

   printTotals(totalRate, totalQuantity, totalGross, totalTax, totalNet, type);
```

```
      printAverages(totalRate, totalQuantity, totalGross, totalTax, totalNet, counter, type);
   }

   // (+) static void printTotals(double totalRate, double totalQuantity, double totalGross, double totalTax, 
double totalNet, char type)
   public static void printTotals(double totalRate, double totalQuantity, double totalGross, double totalTax, 
double totalNet, char type)
   {
      printString38("Totals: ");
      if (type != 's')
      {
         printDouble12(totalRate);
         printDouble12(totalQuantity);
      }
      else
      {
         printString12("");
         printString12("");
      }
      printDouble12(totalGross);
      printDouble12(totalTax);
      printDouble12(totalNet);
      System.out.println();
   }

   // (+) static void printAverages(double totalRate, double totalQuantity, double totalGross, double 
totalTax, double totalNet, int empNum, char type)
   public static void printAverages(double totalRate, double totalQuantity, double totalGross, double 
totalTax, double totalNet, int empNum, char type)
   {
      printString38("Averages: ");
      if (type != 's')
      {
         printDouble12(totalRate/empNum);
         printDouble12(totalQuantity/empNum);
      }
      else
      {
         printString12("");
         printString12("");
      }
      printDouble12(totalGross/empNum);
      printDouble12(totalTax/empNum);
      printDouble12(totalNet/empNum);
      System.out.println();
   }


   // (+) static void printFooter(Employees myEmps)
   public static void printFooter(Employees myEmps)
   {
      double totalGrossPay = 0;
      double totalTaxAmt = 0;
      double totalNetPay = 0;

      while(myEmps.hasNext())
      {
         EmployeeRecord tempRecord = new EmployeeRecord(myEmps.iterate());
         totalGrossPay += tempRecord.grossPay;
         totalTaxAmt += tempRecord.taxAmt;
         totalNetPay += tempRecord.netPay;
      }

      printString38("Grand Totals:");
      printString12("");
      printString12("");
      printDouble12(totalGrossPay);
      printDouble12(totalTaxAmt);
```

```java
      printDouble12(totalNetPay);
      System.out.println();

      printString38("Grand Averages:");
      printString12("");
      printString12("");
      printDouble12(totalGrossPay/myEmps.getLength());
      printDouble12(totalTaxAmt/myEmps.getLength());
      printDouble12(totalNetPay/myEmps.getLength());
      System.out.println();
   }

   // (+) static void printString38(String str)
   public static void printString38(String str)
   {
      System.out.printf("%-38s", str);
   }

   // (+) static void printString12(String str)
   public static void printString12(String str)
   {
      System.out.printf("%12s", str);
   }

   // (+) static void printDouble12(double value)
   public static void printDouble12(double value)
   {
      System.out.printf("%12.2f", value);
   }

   // (+) static String concatenateName(String lastName, String firstName)
   public static String concatenateName(String lastName, String firstName)
   {
      return lastName + ", " + firstName;
   }
}
```

Output Screenshot



```
==================================================================================================
                                  YOUR FINANCIAL REPORT ANALYSIS
==================================================================================================

Employee (Hourly)                              Pay       Hours       Gross         Tax         Net
Name                                          Rate      Worked         Pay      Amount         Pay
=======                                      =====       =====       =====       =====       =====
Davidson, Carl                                8.75       38.00      332.50       49.88      282.63
Doe, John                                    17.00       46.50      845.75      126.86      718.89

Totals:                                      25.75       84.50     1178.25      176.74     1001.51
Averages:                                    12.88       42.25      589.13       88.37      500.76

Employee (Salary)                                                   Salary         Tax         Net
Name                                                                   Pay      Amount         Pay
==============================================================       =====       =====       =====
Prentiss, Paula                                                     795.38      119.31      676.07

Totals:                                                             795.38      119.31      676.07
Averages:                                                           795.38      119.31      676.07

Employee (PieceWork)                        Pieces    Price of       Gross         Tax         Net
Name                                          Sold       Piece         Pay      Amount         Pay
=======                                      =====       =====       =====       =====       =====
Marion, Louise                               13.00       40.00      520.00       78.00      442.00
Whittle, Ed                                  11.50       25.00      287.50       43.13      244.38

Totals:                                      24.50       65.00      807.50      121.13      686.38
Averages:                                    12.25       32.50      403.75       60.56      343.19

Grand Totals:                                                      2781.13      417.17     2363.96
Grand Averages:                                                     556.23       83.43      472.79
```