Marco Martinez

CISP 401

Assignment 4

**Hierarchy Chart**

*Classes*  
 EmployeeRecord  
 Employee  
 Hourly  
 Salary  
 Piece

IntegerDataItem

StringDataItem  
 GenericItemType  
 GenericContainer  
 AppDriver  
 *Associations* Employee(1) --- inherits --- (1) GenericItemType  
 IntegerDataItem(1) --- inherits --- (1) GenericItemType  
 StringDataItem(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainerClasses  
 *GenericContainer*  
 CLASS CONSTRUCTOR  
 (+) GenericContainer()  
 (+) GenericContainer(int size)  
 (+) GenericContainer(GenericContainer gc)  
   
 CHANGE STATE SERVICES  
 (+) void init()  
 (+) void add(GenericItemType git)  
 (+) void remove(GenericItemType git)  
 (+) GenericItemType search(GenericItemType key)  
 (-) GenericItemType biSearch(GenericItemType key,int low,int high)  
 (+) void sort()  
 (-) void qSort(int start, int finish)  
 (+) void Iterator\_Initialize()

READ STATE SERVICES  
 (+) int getMax()  
 (+) int getLength()  
 (+) int getCurrentIndex()  
 (+) GenericItemType get(int i)  
 (+) boolean Iterator\_hasNext()  
 (+) GenericItemType Iterator\_getNext()

*GenericItemType*  
 (+) Abstract boolean isLess(GenericItemType)  
 (+) Abstract boolean isEqual(GenericItemType)  
 (+) Abstract boolean isGreater(GenericItemType)

*IntegerDataItem*  
 INSTANCE VARIABLE DECLARATION  
 (-) int myValue;  
   
 CLASS CONSTRUCTORS  
 (+) IntegerDataItem()  
 (+) IntegerDataItem(int i)  
 (+) IntegerDataItem(IntegerDataItem idi)  
   
 CHANGE STATE SERVICES  
 (+) void set(int i)  
   
 READ STATE SERVICES  
 (+) boolean isLess(GenericItemType git)  
 (+) boolean isEqual(GenericItemType git)  
 (+) boolean isGreater(GenericItemType git)  
 (+) int get()  
 (+) String toString()

*StringDataItem*  
 INSTANCE VARIABLE DECLARATION  
 (-) String myString;  
   
 CLASS CONSTRUCTORS  
 (+) StringDataItem()  
 (+) StringDataItem(String s)  
 (+) StringDataItem(StringDataItem sdi)  
   
 CHANGE STATE SERVICES  
 (+) void set(String s)  
   
 READ STATE SERVICES  
 (+) boolean isLess(GenericItemType git)  
 (+) boolean isEqual(GenericItemType git)  
 (+) boolean isGreater(GenericItemType git)  
 (+) String get()  
 (+) 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  
 (+) boolean isLess(GenericItemType git)  
 (+) boolean isEqual(GenericItemType git)  
 (+) boolean isGreater(GenericItemType git)  
 (+) EmployeeRecord get()  
 (+) String toString()

*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()

*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()

*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()

*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()

**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)

*GenericContainer*

GenericContainer() 🡪 s0

GenericContainer(Int) 🡪 s0

GenericContainer(GenericContainer) 🡪 s0

S0 🡪 init() 🡪 s0

S0 🡪 add(GenericItemType) 🡪 s0

S0 🡪 remove(GenericItemType) 🡪 s0

S0 🡪 search(GenericItermType) 🡪 s(Err)

S0 🡪 sort() 🡪 s1

S0 🡪 getMax() 🡪 s(Terminal)

S0 🡪 getLength() 🡪 s(Terminal)

S0 🡪 getCurrentIndex() 🡪 s(Terminal)

S0 🡪 get(int) 🡪 s(Terminal)

S0 🡪 Iterator\_Initialize() 🡪 s0

S0 🡪 Iterator\_hasNext() 🡪 s(Terminal)

S0 🡪 Iterator\_getNext() 🡪 s(Terminal)

S1 🡪 init() 🡪 s0

S1 🡪 add(GenericItemType) 🡪 s0

S1 🡪 remove(GenericItemType) 🡪 s0

S1 🡪 search(GenericItermType) 🡪 s(Err)

S1🡪 sort() 🡪 s1

S1 🡪 getMax() 🡪 s(Terminal)

S1 🡪 getLength() 🡪 s(Terminal)

S1 🡪 getCurrentIndex() 🡪 s(Terminal)

S1 🡪 get(int) 🡪 s(Terminal)

S1 🡪 Iterator\_Initialize() 🡪 s0

S1 🡪 Iterator\_hasNext() 🡪 s(Terminal)

S1 🡪 Iterator\_getNext() 🡪 s(Terminal)

**Use Case Scenario (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 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 3:*

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 4:*

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 5:*

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 6:*

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 7:*

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 8:*

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 9:*

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 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 redundant grossPay calculation.

5. User requests the processed values via get() method.

6. User exits application.

*Normal Scenario 11:*

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.

*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.

**Java Source Code**

/\*\*   
 @author Marco Martinez  
 @fileName GenericContainer.java  
 @version 1.0  
 @description This program will construct and manipulate GenericContainer objects.  
   
 Classes  
 EmployeeRecord  
 Employee  
 Hourly  
 Salary  
 Piece  
 GenericItemType  
 GenericContainer  
 AppDriver  
   
 Associations  
 Employee(1) --- inherits --- (1) GenericItemType  
 IntegerDataItem(1) --- inherits --- (1) GenericItemType  
 StringDataItem(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainerClasses  
   
 GenericContainer  
 CLASS CONSTRUCTOR  
 (+) GenericContainer()  
 (+) GenericContainer(int size)  
 (+) GenericContainer(GenericContainer gc)  
   
 CHANGE STATE SERVICES  
 (+) void init()  
 (+) void add(GenericItemType git)]  
 (+) void remove(GenericItemType git)  
 (+) GenericItemType search(GenericItemType key)  
 (-) GenericItemType biSearch(GenericItemType key,int low,int high)  
 (+) void sort()  
 (-) void qSort(int start, int finish)  
 (+) void Iterator\_Initialize()  
 (+) boolean Iterator\_hasNext()  
 (+) GenericItemType Iterator\_getNext()  
   
 @date 12/12/2018  
  
 Program Change Log   
 ==========================  
 Name Date Description  
 Marco 12/12 Create baseline for GenericContainer.  
 \*/  
   
public class GenericContainer  
{  
 // INSTANCE VARIABLE DECLARATION  
 private final int MAXSIZE = 30;  
 private int sizeLimit,  
 index,  
 currentIndex;  
 private GenericItemType[] collection;  
   
 // CLASS CONSTRUCTORS  
 // (+) GenericContainer()  
 public GenericContainer()  
 {  
 this.collection = new GenericItemType[MAXSIZE];  
 this.sizeLimit = MAXSIZE;   
 this.currentIndex = 0;   
 }  
   
 // (+) GenericContainer(int size)  
 public GenericContainer(int size)  
 {   
 this.currentIndex = 0;  
 if (size <= MAXSIZE)  
 this.sizeLimit = size;  
 else  
 this.sizeLimit = MAXSIZE;  
 }  
   
 // (+) GenericContainer(GenericContainer gc)  
 public GenericContainer(GenericContainer gc)  
 {   
 this.currentIndex = this.index = 0;  
 gc.Iterator\_Initialize();  
 while (gc.Iterator\_hasNext())   
 {  
 this.collection[this.currentIndex] = gc.Iterator\_getNext();  
 this.index++;  
 }  
 }  
   
 // CHANGE STATE SERVICES   
 // (+) void init()  
 public void init()  
 {   
 Iterator\_Initialize();  
 while (Iterator\_hasNext())   
 this.collection[this.currentIndex] = null;  
 }  
   
 // (+) void add(GenericItemType git)  
 public void add(GenericItemType git)  
 {  
 this.collection[this.index++] = git;  
 }  
   
 // (+) void remove(GenericItemType git)  
 public void remove(GenericItemType git)  
 {  
 Iterator\_Initialize();  
 GenericItemType temp;  
 while (Iterator\_hasNext())  
 {  
 temp = Iterator\_getNext();  
 if (temp.isEqual(git))  
 {  
 this.collection[this.currentIndex-1] = this.collection[this.index-1];  
 this.collection[this.index-1] = new IntegerDataItem();  
 this.index--;  
 return;  
 }  
 }  
 }  
   
 // (+) GenericItemType search(GenericItemType key)  
 public GenericItemType search(GenericItemType key)  
 {  
 return biSearch(key,0,this.index);  
 }  
   
 // (-) GenericItemType biSearch(GenericItemType key,int low,int high)  
 private GenericItemType biSearch(GenericItemType key,int low,int high)  
 {   
 while(high >= low)   
 {  
 int middle = (low + high) / 2;  
 if (collection[middle].isEqual(key))  
 {  
 return collection[middle];  
 }  
 if (collection[middle].isGreater(key))   
 {  
 return biSearch(key, low, middle-1);  
 }  
 if (collection[middle].isLess(key))   
 {  
 return biSearch(key, middle+1, high);  
 }   
 }   
 return new StringDataItem();  
 }  
   
 // (+) 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;  
 GenericItemType pivot = this.collection[start + (finish - start) / 2];  
  
 while (i <= j)   
 {  
 while (this.collection[i].isLess(pivot))   
 {  
 i++;  
 }  
  
 while (this.collection[j].isGreater(pivot))   
 {  
 j--;  
 }  
  
 if (i <= j)   
 {  
 GenericItemType temp = this.collection[i];  
 this.collection[i] = this.collection[j];  
 this.collection[j] = temp;  
 i++;  
 j--;  
 }  
 }  
   
 if (start < j)   
 {  
 qSort(start, j);  
 }  
 if (i < finish)   
 {  
 qSort(i, finish);  
 }  
 }  
   
 // (+) int getMax()  
 public int getMax()  
 {  
 if (this.sizeLimit != 0)  
 return sizeLimit;  
 else  
 return MAXSIZE;  
 }  
   
 // (+) int getLength()  
 public int getLength()  
 {  
 return this.index;  
 }  
   
 // (+) int getCurrentIndex()  
 public int getCurrentIndex()  
 {  
 return this.currentIndex;  
 }  
   
 // (+) GenericItemType get(int i)  
 public GenericItemType get(int i)  
 {  
 return this.collection[i];  
 }  
   
 // (+) void Iterator\_Initialize()  
 public void Iterator\_Initialize()   
 {  
 this.currentIndex = 0;  
 }  
   
 // (+) boolean Iterator\_hasNext()  
 public boolean Iterator\_hasNext()   
 {  
 return this.currentIndex <= this.index-1;  
 }  
   
 // (+) GenericItemType Iterator\_getNext()  
 public GenericItemType Iterator\_getNext()  
 {   
 return this.collection[this.currentIndex++];  
 }  
}

/\*\*   
 @author Marco Martinez  
 @fileName GenericItemType.java  
 @version 1.0  
 @description This program will construct and manipulate GenericItemType objects.  
   
 Classes  
 GenericItemType  
 IntegerDataType  
 StringDataType  
 GenericContainer  
 AppDriver  
   
 Associations  
 IntegerDataType --- 1 : 1 (inherits) ---> GenericItempType  
 StringDataType --- 1 : 1 (inherits) ---> GenericItemType  
 GenericContainer --- 1 : m (contains) ---> GenericItemType  
 AppDriver --- 1 : 1 (uses) ---> GenericContainer  
   
 GenericItemType // is an abstract class  
 (+) Abstract boolean isLess(GenericItemType)  
 (+) Abstract boolean isEqual(GenericItemType)  
 (+) Abstract boolean isGreater(GenericItemType)  
  
   
 @date 10/11/2018  
  
 Program Change Log   
 ==========================  
 Name Date Description  
 Marco 12/12 Create baseline for GenericItemType.  
 \*/  
   
public abstract class GenericItemType  
{   
 // (+) abstract boolean isLess(GenericItemType git)  
 public abstract boolean isLess(GenericItemType git);  
   
 // (+) abstract boolean isEqual(GenericItemType git)  
 public abstract boolean isEqual(GenericItemType git);  
   
 // (+) abstract boolean isGreater(GenericItemType git)  
 public abstract boolean isGreater(GenericItemType git);  
}

/\*\*   
 @author Marco Martinez  
 @fileName GenericItemType.java  
 @version 1.0  
 @description This program will construct and manipulate IntegerDataType objects.  
   
 Classes  
 GenericItemType  
 IntegerDataType  
 StringDataType  
 GenericContainer  
 AppDriver  
   
 Associations  
 IntegerDataType --- 1 : 1 (inherits) ---> GenericItempType  
 StringDataType --- 1 : 1 (inherits) ---> GenericItemType  
 GenericContainer --- 1 : m (contains) ---> GenericItemType  
 AppDriver --- 1 : 1 (uses) ---> GenericContainer  
   
 IntegerDataItem  
 INSTANCE VARIABLE DECLARATION  
 (-) int myValue;  
   
 CLASS CONSTRUCTORS  
 (+) IntegerDataItem()  
 (+) IntegerDataItem(int i)  
 (+) IntegerDataItem(IntegerDataItem idi)  
   
 CHANGE STATE SERVICES  
 (+) void set(int i)  
   
 READ STATE SERVICES  
 (+) boolean isLess(GenericItemType git)  
 (+) boolean isEqual(GenericItemType git)  
 (+) boolean isGreater(GenericItemType git)  
 (+) int get()  
 (+) String toString()  
  
   
 @date 12/12/2018  
  
 Program Change Log   
 ==========================  
 Name Date Description  
 Marco 12/12 Create baseline for IntegerDataType.  
 \*/  
   
public class IntegerDataItem extends GenericItemType  
{  
 // INSTANCE VARIABLE DECLARATION  
 private int myValue;  
   
 //CLASS CONSTRUCTORS  
 // (+) IntegerDataItem()  
 public IntegerDataItem(){}  
   
 // (+) IntegerDataItem(int i)  
 public IntegerDataItem(int i)  
 {   
 set(i);  
 }  
   
 // (+) IntegerDataItem(IntegerDataItem idi)  
 public IntegerDataItem(IntegerDataItem idi)  
 {   
 set(idi.get());  
 }  
   
 // CHANGE STATE SERVICES  
 // (+) void set(int i)  
 public void set(int i)   
 {   
 myValue = i;  
 }  
   
 // READ STATE SERVICES  
 // (+) boolean isLess(GenericItemType git)  
 public boolean isLess(GenericItemType git)  
 {   
 return (myValue < ((IntegerDataItem) git).get());  
 }  
   
 // (+) boolean isEqual(GenericItemType git)  
 public boolean isEqual(GenericItemType git)  
 {   
 return (myValue == ((IntegerDataItem) git).get());  
 }  
   
 // (+) boolean isGreater(GenericItemType git)  
 public boolean isGreater(GenericItemType git)  
 {   
 return (myValue > ((IntegerDataItem) git).get());  
 }  
   
 // (+) int get()  
 public int get()   
 {   
 return myValue;  
 }  
   
 // (+) String toString()  
 public String toString()   
 {  
 return Integer.toString(myValue);  
 }  
}

/\*\*   
 @author Marco Martinez  
 @fileName StringDataItem.java  
 @version 1.0  
 @description This program will construct and manipulate StringDataItem objects.  
   
 Classes  
 GenericItemType  
 IntegerDataType  
 StringDataType  
 GenericContainer  
 AppDriver  
   
 Associations  
 IntegerDataType --- 1 : 1 (inherits) ---> GenericItempType  
 StringDataType --- 1 : 1 (inherits) ---> GenericItemType  
 GenericContainer --- 1 : m (contains) ---> GenericItemType  
 AppDriver --- 1 : 1 (uses) ---> GenericContainer  
   
 StringDataItem  
 INSTANCE VARIABLE DECLARATION  
 (-) String myString;  
   
 CLASS CONSTRUCTORS  
 (+) StringDataItem()  
 (+) StringDataItem(String s)  
 (+) StringDataItem(StringDataItem sdi)  
   
 CHANGE STATE SERVICES  
 (+) void set(String s)  
   
 READ STATE SERVICES  
 (+) boolean isLess(GenericItemType git)  
 (+) boolean isEqual(GenericItemType git)  
 (+) boolean isGreater(GenericItemType git)  
 (+) String get()  
 (+) String toString()  
   
 @date 12/12/2018  
  
 Program Change Log   
 ==========================  
 Name Date Description  
 Marco 12/12 Create baseline for StringDataItem.  
 \*/  
   
public class StringDataItem extends GenericItemType  
{  
 // INSTANCE VARIABLE DECLARATION  
 private String myString;  
   
 // CLASS CONSTRUCTORS  
 // (+) StringDataItem()  
 public StringDataItem(){}  
   
 // (+) StringDataItem(String s)  
 public StringDataItem(String s)  
 {   
 myString = new String(s);  
 }  
   
 // (+) StringDataItem(StringDataItem sdi)  
 public StringDataItem(StringDataItem sdi)  
 {   
 set(sdi.get());  
 }  
   
 // CHANGE STATE SERVICES  
 // (+) void set(String s)  
 public void set(String s)   
 {   
 myString = s;  
 }  
   
 // READ STATE SERVICES  
 // (+) boolean isLess(GenericItemType git)  
 public boolean isLess(GenericItemType git)  
 {   
 return ( myString.compareTo(((StringDataItem) git).get()) < 0);  
 }  
   
 // (+) boolean isEqual(GenericItemType git)  
 public boolean isEqual(GenericItemType git)  
 {   
 return ( myString.compareTo(((StringDataItem) git).get()) == 0);  
 }  
   
 // (+) boolean isGreater(GenericItemType git)  
 public boolean isGreater(GenericItemType git)  
 {   
 return ( myString.compareTo(((StringDataItem) git).get()) > 0);  
 }  
   
 // (+) String get()  
 public String get()   
 {   
 return myString;  
 }  
   
 // (+) String toString()  
 public String toString()  
 {  
 return "Value of myString: " + myString;  
 }  
}

/\*\*   
 @author Marco Martinez  
 @fileName EmployeeRecord.java  
 @version 1.1  
 @description This program will construct and manipulate EmployeeRecord objects.  
   
 Classes  
 EmployeeRecord  
 Employee  
 Hourly  
 Salary  
 Piece  
 GenericItemType  
 GenericContainer  
 AppDriver  
   
 Associations  
 Employee(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainer  
   
 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  
 GenericItemType  
 GenericContainer  
 AppDriver  
   
 Associations  
 Employee(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainer  
   
 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 extends GenericItemType  
{  
 // 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  
 // (+) boolean isLess(GenericItemType git)  
 public boolean isLess(GenericItemType git)  
 {  
 if(this.e.lastName.compareToIgnoreCase(((Employee)(git)).get().lastName) < 0)  
 return true;  
 else  
 return false;  
 }  
   
 // (+) boolean isEqual(GenericItemType git)  
 public boolean isEqual(GenericItemType git)  
 {  
 if(this.e.lastName.compareToIgnoreCase(((Employee)(git)).get().lastName) == 0)  
 return true;  
 else  
 return false;  
 }  
   
 // (+) boolean isGreater(GenericItemType git)  
 public boolean isGreater(GenericItemType git)  
 {  
 if(this.e.lastName.compareToIgnoreCase(((Employee)(git)).get().lastName) > 0)  
 return true;  
 else  
 return false;  
 }  
   
 // (+) EmployeeRecord get()  
 public EmployeeRecord get()  
 {  
 return this.e;  
 }  
   
 // (+) String toString()  
 public String toString()  
 {  
 return this.e.toString();  
 }  
}

/\*\*   
 @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  
 GenericItemType  
 GenericContainer  
 AppDriver  
   
 Associations  
 Employee(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainer  
   
 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;  
 }  
}

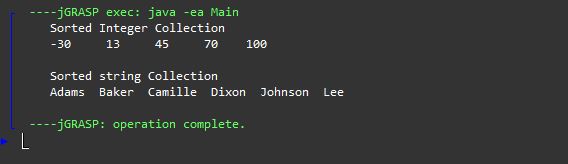
/\*\*   
 @author Marco Martinez  
 @fileName Piece.java  
 @version 1.0  
 @description This program will construct and manipulate Piece-Employee objects.  
   
 Classes  
 EmployeeRecord  
 Employee  
 Hourly  
 Salary  
 Piece  
 GenericItemType  
 GenericContainer  
 AppDriver  
   
 Associations  
 Employee(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainer  
   
 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 Salary.java  
 @version 1.0  
 @description This program will construct and manipulate Salary-Employee objects.  
   
 Classes  
 EmployeeRecord  
 Employee  
 Hourly  
 Salary  
 Piece  
 GenericItemType  
 GenericContainer  
 AppDriver  
   
 Associations  
 Employee(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainer  
   
 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;  
 }  
}

/\*\*  
 GenericItemType // is an abstract class  
 (+) abstract boolean isLess(GenericItemType)  
 (+) abstract boolean isEqual(GenericItemType)  
 (+) abstract boolean isGreater(GenericItemType)  
   
 IntegerDataType --- 1 : 1 (inherits) ---> GenericItemType  
 (+) all constructors  
 (+) boolean isLess(GenericItemType) // overrides of base method  
 (+) boolean isEqual(GenericItemType)  
 (+) boolean isGreater(GenericItemType)  
 (+) accessors (get(), toString())  
 (+) manipulators  
   
 StringDataType --- 1 : 1 (inherits) ---> GenericItemType  
 (+) boolean isLess(GenericItemType) // override of base method  
 (+) boolean isEqual(GenericItemType)  
 (+) boolean isGreater(GenericItemType)  
   
 GenericContainer --- 1 : m (contains) --- GenericItemType   
\*/  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 GenericContainer gC = new GenericContainer();  
   
 gC.add(new IntegerDataItem(13));  
 gC.add(new IntegerDataItem(-30));   
 gC.add(new IntegerDataItem(100));  
 gC.add(new IntegerDataItem(70));  
 gC.add(new IntegerDataItem(45));  
 gC.sort();  
 System.out.printf(" Sorted Integer Collection\n");  
 gC.Iterator\_Initialize();  
 while (gC.Iterator\_hasNext()) {  
 IntegerDataItem nextOne = (IntegerDataItem )(gC.Iterator\_getNext());   
 System.out.printf(" %5d", nextOne.get());  
 if (!(gC.Iterator\_hasNext())) System.out.printf("\n\n");  
 }  
 GenericContainer sgC= new GenericContainer();  
 sgC.add(new StringDataItem("Johnson"));  
 sgC.add(new StringDataItem("Dixon"));   
 sgC.add(new StringDataItem("Adams"));  
 sgC.add(new StringDataItem("Baker"));  
 sgC.add(new StringDataItem("Lee"));  
 sgC.add(new StringDataItem("Camille"));  
 sgC.sort();  
 System.out.printf(" Sorted string Collection\n");  
 System.out.print(" ");  
 sgC.Iterator\_Initialize();  
 while (sgC.Iterator\_hasNext()) {  
 StringDataItem nextOne = (StringDataItem) (sgC.Iterator\_getNext());   
 System.out.printf(" %s", nextOne.get());  
 if (!(sgC.Iterator\_hasNext())) System.out.printf("\n");  
 }  
 } // main  
} // class

/\*\*   
 @author Marco Martinez  
 @fileName AppDriver.java  
 @version 1.0  
 @description This program will utilize GenericContainer objects for creating a report.  
   
 Classes  
 EmployeeRecord  
 Employee  
 Hourly  
 Salary  
 Piece  
 GenericItemType  
 GenericContainer  
 AppDriver  
   
 Associations  
 Employee(1) --- inherits --- (1) GenericItemType  
 Hourly(1) --- inherits --- (1) Employee  
 Salary(1) --- inherits --- (1) Employee  
 Piece(1) --- inherits --- (1) Employee  
 GenericContainer(1) --- contains --- (m) GenericItemType  
 AppDriver(1) --- uses --- (1) GenericContainer  
   
 AppDriver Class  
 (+) static void getEmployees(GenericContainer myEmps)  
 (+) static void getEmployee(GenericContainer 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(GenericContainer myEmps)  
 (+) static void printHeading()  
 (+) static void printLabels()  
 (+) static void printEmployees(GenericContainer myEmps)  
 (+) static boolean determineIfTypeExists(GenericContainer, char type)  
 (+) static void printEmployee(EmployeeRecord emp, String str, char type)  
 (+) static void printTypeFooter(GenericContainer 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(GenericContainer 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.  
 Marco 12/13 Adjusted for generics.  
 \*/  
// 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  
 GenericContainer myEmps = new GenericContainer();  
   
 // CALLS  
 getEmployees(myEmps);  
 myEmps.sort();  
 printReport(myEmps);  
 }  
   
 // METHODS  
 // (+) static void getEmployees(GenericContainer myEmps)  
 public static void getEmployees(GenericContainer 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.getMax() - 1)   
 {  
 System.out.println("You have hit the maximum amount of employees to enter.");  
 c = 'n';  
 }  
 }  
 input.close();  
 }  
   
 // (+) static void getEmployee(GenericContainer myEmps, String payPrompt, String numOfPrompt, char c, Scanner input)  
 public static void getEmployee(GenericContainer 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(GenericContainer myEmps)  
 public static void printReport(GenericContainer 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(GenericContainer myEmps)  
 public static void printEmployees(GenericContainer 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.Iterator\_hasNext())   
 {  
 emp = new EmployeeRecord(((Employee) myEmps.Iterator\_getNext()).get());  
 if (emp.type == 'h')  
 {  
 temp = new Hourly((Employee) myEmps.get(myEmps.getCurrentIndex()-1));  
 printEmployee(emp,'h',((Hourly) temp).getRate(),((Hourly) temp).getHours());  
 }  
 }  
 myEmps.Iterator\_Initialize();  
 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.Iterator\_hasNext())   
 {  
 emp = new EmployeeRecord(((Employee) myEmps.Iterator\_getNext()).get());  
 if (emp.type == 's')  
 {  
 temp = new Salary((Employee) myEmps.get(myEmps.getCurrentIndex()-1));  
 printEmployee(emp,'s',((Salary) temp).getSalary(), 0.00);  
 }  
 }  
 myEmps.Iterator\_Initialize();  
 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.Iterator\_hasNext())   
 {  
 emp = new EmployeeRecord(((Employee)myEmps.Iterator\_getNext()).get());  
 if (emp.type == 'p')  
 {  
 temp = new Piece(((Employee)myEmps.get(myEmps.getCurrentIndex()-1)));  
 printEmployee(emp,'p',((Piece) temp).getPrice(),((Piece) temp).getPieces());  
 }  
 }  
 myEmps.Iterator\_Initialize();  
 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(GenericContainer myEmps, char type)  
 public static boolean determineIfTypeExists(GenericContainer myEmps, char type)  
 {  
 EmployeeRecord emp;  
   
 while(myEmps.Iterator\_hasNext())  
 {  
 emp = new EmployeeRecord(((Employee)myEmps.Iterator\_getNext()).get());  
 if (emp.type == type)   
 {  
 myEmps.Iterator\_Initialize();  
 return true;  
 }  
 }  
 myEmps.Iterator\_Initialize();  
 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(GenericContainer myEmps, char type)  
 public static void printTypeFooter(GenericContainer myEmps, char type)  
 {  
 double totalRate = 0,  
 totalQuantity = 0,  
 totalGross = 0,  
 totalTax = 0,  
 totalNet = 0;  
 int counter = 0;  
 Employee temp;  
   
 while(myEmps.Iterator\_hasNext())  
 {   
 if ((((Employee)(myEmps.Iterator\_getNext())).get()).type == type)  
 {   
 counter++;  
 switch (type)  
 {  
 case 'h':  
 temp = new Hourly((Employee)myEmps.get(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(((Employee)myEmps.get(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((Employee)myEmps.get(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.Iterator\_Initialize();  
   
 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(GenericContainer myEmps)  
 public static void printFooter(GenericContainer myEmps)  
 {  
 double totalGrossPay = 0;  
 double totalTaxAmt = 0;  
 double totalNetPay = 0;  
   
 while(myEmps.Iterator\_hasNext())  
 {  
 EmployeeRecord tempRecord = new EmployeeRecord(((Employee)myEmps.Iterator\_getNext()).get());  
 totalGrossPay += tempRecord.grossPay;  
 totalTaxAmt += tempRecord.taxAmt;  
 totalNetPay += tempRecord.netPay;  
 }  
   
 printString38("Grand Totals:");  
 printString12("");  
 printString12("");  
 printDouble12(totalGrossPay);  
 printDouble12(totalTaxAmt);  
 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;  
 }  
}

**Pre-defined Datatype Screenshot**



**Programmer-defined Datatype Screenshot**

