Name(s):_____

Problem practice – Inheritance through Abstract Classes

Fill in the chart below with T or F depending on whether or not the characteristic on the left can be applied to the class type or interface on the top.

characteristic	(non-abstract) super class	sub class	abstract class
May **only** have abstract methods.	F	F	F
May have one or more abstract methods	F	F	T
May have instance variables	T	T	T
May have class variables	T	T	T
May have class constants	T	T	T
May have concrete methods	T	T	T
May be extended by another class	T	T	T
May be used to declare object reference variables	T	T	F

Given the TwoPartMeasure, Weight, and Length classes from the lab and assuming the following code, indicate which of the statements if present in main will compile (C) or not compile (X). Indicate in the space underneath the reason for any statement that will not compile.

```
/**
 * A driver that can be used to test
 * some aspects of the Length and Weight classes
 *
 * @author  Nancy Harris
 * @version 1.0
 */
public class Driver3
{
    /**
     * The entry point of the application
     *
     * @param args   The command line arguments
     */
    public static void main(String[] args)
    {
      TwoPartMeasure       tpm;
      TwoPartMeasure[]     tpmArray;
      Length               myLength;
      Weight               myWeight;
       int                       units;

      tpmArray = new TwoPartMeasure[3];
       myLength = new Length(12, 4);
      myWeight = new Weight(125, 3);

      // ... more code goes here
   }
}
```

C or X	Statement in main
C	1. tpm = myLength;
X	2. tpm = new TwoPartMeasure(5, 3);
X	3. myLength = myWeight;
X	4. myLength = tpm;
C	5. tpmArray[0] = myWeight;
C	6. tpmArray[1] = new Length(3, 4);
X	7. tpmArray[2] = new TwoPartMeasure(5, 2, true);
C	8. units = myWeight.smallsPerLarge;
C	9. myWeight.initializeUnits();
X	10. units = myLength.toSmall();
C	11. myWeight.equals(myLength);
C	12. System.out.println(myWeight);
C	13. myWeight.changeBy(myLength);

Statement	Rationale
2.	TwoPartMeasure is abstract so cannot be instantiated.
3.	Incompatible types.
4.	Tpm has not been assigned a value. Even if it did have a value, it is not possible to assign from a variable of type TwoPartMeasure to a variable of type Length without a cast.
7.	TwoPartMeasure is abstract so cannot be instantiated.
10.	toSmall() is a private method of TwoPartMeasure.

Given the TwoPartMeasure, Weight, and Length classes, answer the following:

1. Which **data** members of TwoPartMeasure are “visible” in Weight?

```
smallsPerLarge, largeUnitsSingular, largeUnitsPlural, smallUnitsSingular, smallUnitsPlural
```

2. Which **method** members of TwoPartMeasure are “visible” in Weight?

All except `toSmall()`.

3. Which **method** members of Weight are “visible” in TwoPartMeasure?

All are visible from TwoPartMeasure, in the same way they would be visible to any other class in the same package.

4. If we changed the default constructor of Weight from what is listed in the reference to this, would this class compile? Why or why not?

```
/**
 * Default Constructor
 */
  public Weight()
  {
     this(0, 0, true);
  }
```

It would compile. The original constructor calls the three-argument superclass constructor directly. This constructor calls the three-argument length constructor, which then calls the three-argument superclass constructor.

5. If we wanted to create accessor methods for the large and small units, where should they go (TwoPartMeasure, Weight, Length or some combination)? What would one of the methods headers look like (large or small)?

It makes the most sense to put them in TwoPartMeasure:

```
    public String getLargeUnitsSingular()
    {
          return largeUnitsSingular;
    }

    public String getSmallUnitsSingular()
    {
          return smallUnitsSingular;
    }
```

```
public abstract class TwoPartMeasure
   {
      private int           large, sign, small;

      protected int         smallsPerLarge;
      protected String      largeUnitsSingular, largeUnitsPlural;
      protected String      smallUnitsSingular, smallUnitsPlural;

      public TwoPartMeasure()
      {
         this(0, 0, true);
      }
      public TwoPartMeasure(int large, int small)
      {
         this(large, small, true);
      }
      public TwoPartMeasure(int large, int small, boolean positive)
      {
         this.large = Math.abs(large);
         this.small = Math.abs(small);

        this.sign   = 1;
         if (!positive) this.sign = -1;


         initializeUnits();
      }
      public void changeBy(TwoPartMeasure other)
      {
         int           otherTotal, thisTotal, total;

         otherTotal = other.toSmall();
         thisTotal  = this.toSmall();

         total = thisTotal + otherTotal;

         large = total / smallsPerLarge;
         small = total % smallsPerLarge;
      }
      public boolean equals(TwoPartMeasure other)
      {
         boolean comparison;
         int     otherTotal, thisTotal;

         thisTotal  = this.toSmall();
         otherTotal = other.toSmall();

         comparison = false;
         if (thisTotal == otherTotal) comparison = true;

         return comparison;
      }

      protected abstract void initializeUnits();

      private int toSmall()
      {
         int         total;

         total = sign * ((large * smallsPerLarge) + small);
         return total;
      }

      public String toString()
      {
         String     s;

         s = new String();
         if (sign < 0) s += "Negative ";

         if (large == 1) s += large + " " + largeUnitsSingular;
         else            s += large + " " + largeUnitsPlural;

         if (small == 1) s += "  " + small + " " + smallUnitsSingular;
         else            s += "  " + small + " " + smallUnitsPlural;
```

```
         return s;
      }

   }
```

```
public class Length extends TwoPartMeasure
   {

      public Length()
      {
         super(0, 0, true);
      }


      public Length(int feet, int inches)
      {
         super(feet, inches, true);
      }


      public Length(int feet, int inches, boolean positive)
      {
         super(feet, inches, positive);
      }
      protected void initializeUnits()
      {
         smallsPerLarge = 12;

         largeUnitsSingular = "foot";
         largeUnitsPlural   = "feet";

         smallUnitsSingular = "inch";
         smallUnitsPlural   = "inches";
      }
   }
```

```
   public class Weight extends TwoPartMeasure
   {

      public Weight()
      {
         super(0, 0, true);
      }

      public Weight(int pounds, int ounces)
      {
         super(pounds, ounces, true);
      }

      public Weight(int pounds, int ounces, boolean positive)
      {
         super(pounds, ounces, positive);
      }
      protected void initializeUnits()
      {
         smallsPerLarge = 16;

         largeUnitsSingular = "pound";
         largeUnitsPlural   = "pounds";

         smallUnitsSingular = "ounce";
         smallUnitsPlural   = "ounces";
      }
   }
```