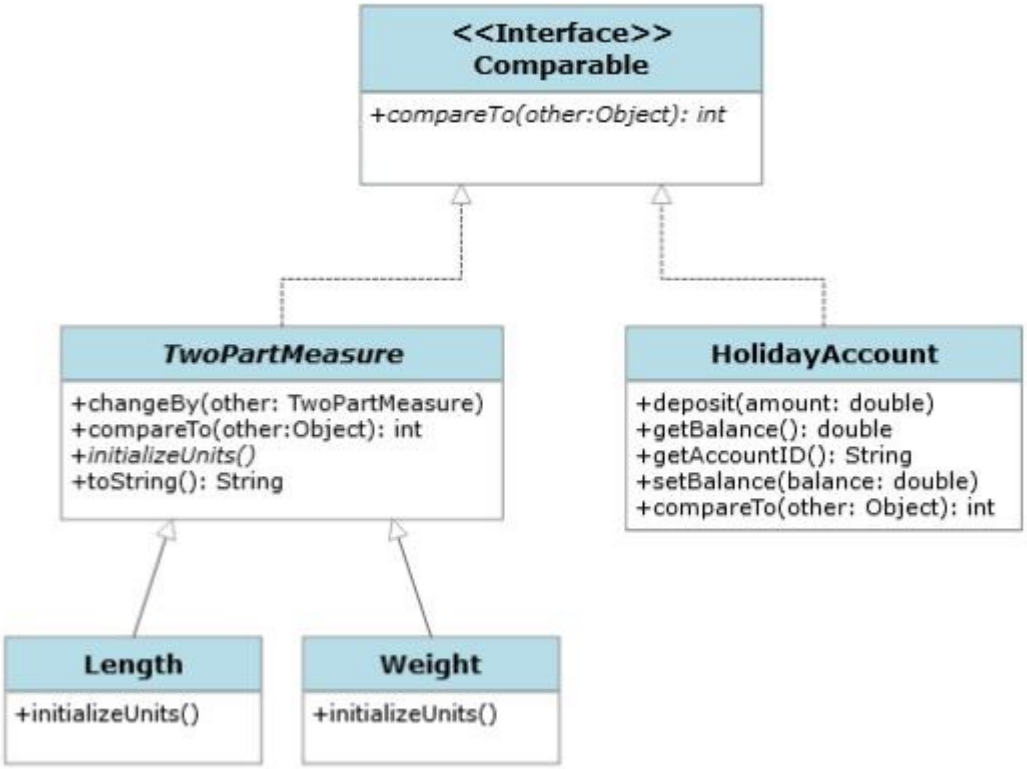
Q1:



Take a moment to familiarize yourself with these classes and interfaces. The UML indicates abstract methods/classes by slanted text, specializations by arrows with solid lines, and realizations/implementations of interfaces by arrows with dashed lines.

Please answer the following questions:

1. What is the benefit of including TwoPartMeasure in this design?

2. What is the purpose of including this abstract method in TwoPartMeasure?

3. Why is it a good idea to include the `Comparable` interface in this design?

4. Is there a reason HolidayAccount doesn't extend TwoPartMeasure?

Answers:

1. It implements methods that are needed in both of its subclasses, Length and Weight. If it were omitted, those two classes would contain duplicate code.

2. There are two reasons. First, by including it as an abstract method the compiler can check to ensure if the method has been included in concrete subclasses. Second, the initializeUnits() method is called in the TwoPartMeasure class' constructor.

3. It allows us to identify which objects can be compared and, hence, whether an array of a particular kind of objects can be sorted.

4. Because there is no sense in which a HolidayAccount "is a" TwoPartMeasure. They have no attributes in common and the only method they have in common is compareTo() (and, perhaps, toString()).

Q2: Given the codes below:

```
class Animal{}
interface Feline{}
class Lion{}
```

If we wanted to link the previous types with inheritance, which of the following snippets is valid from the compiler's point of view (they could all be valid)?

1. class Animal extends Feline {}
2. interface Feline extends Animal {}
3. class Lion extends Feline {}
4. class Lion extends Animal implements Feline {}
5. class Animal extends Lion implements Feline {}

Q3: Given the following hierarchy:

```
interface X {
        void method();
}

interface Y extends X {
}

abstract class Z implements Y {
}

public final class M extends Z {
        public void method() {
        }
}
```

Which of the following statements are false (it is possible to choose more than one statement):

1. Class Z cannot implements an interface.
2. It is not possible to extend interface Y to another interface.
3. Class Z implementing Y also inherits the abstract method of X.

4. Class M cannot be declared final.

Q4: Consider this simple Java program.

It contains multiple errors that prevent it from compiling, running and producing the expected output shown below.

Produce a corrected version of this code - and submit it as a file. For full marks you must include comments explaining each correction that was made.

```
import java.util.InputMismatchException;

  public class SimpleErrors {

    public void main(String[] args) {

      Scanner stdin = new Scanner(System.in)
      System.out.print("Enter two numbers: ");
      try {
        double input1 = stdin.nextInt;
        int input2 = stdin.nextInt;

        int num1 = input1;
        int num2 = in2;

        while (num1 % num2 != 0)
          oldn1 = num1;
          oldn2 = num2;
          num1 = oldn2;
          num2 = oldn1 % oldn2;

        num2 = Math.abs(num2);
      } catch (InputMismatchException) {
        System.err.println("Error: input must be two numbers.");
        System.exit(1);
      }

      System.out.println("The GCD of ",input1," and ",input2" is num2");
    }
  }
```

Example output 1:

```
Enter two numbers: 12 8
The GCD of 12 and 8 is 4
```

Example output 2:

```
Enter two numbers: 8 -12
The GCD of 8 and -12 is 4
```

***Answer*:**

```
import java.util.Scanner;

import java.util.InputMismatchException;
```

```
public class SimpleErrors {


    // Change 'public void main' to 'public static void main'

    public static void main(String[] args) {

        Scanner stdin = new Scanner(System.in); // Missing semicolon here

        System.out.print("Enter two numbers: ");

        try {

            // Change from nextInt to nextDouble for the first input to
capture doubles correctly

            double input1 = stdin.nextDouble(); // Missing method call
parentheses

            int input2 = stdin.nextInt(); // Missing method call
parentheses



            // Proper casting from double to int where necessary and fix
variable names

            int num1 = (int) input1; // Cast double to int

            int num2 = input2; // Correct variable name from in2 to input2



            // Encapsulate the while loop with braces to ensure correct
block execution

            int oldn1, oldn2; // Declare variables before using them

            while (num1 % num2 != 0) {

                oldn1 = num1;

                oldn2 = num2;

                num1 = oldn2;

                num2 = oldn1 % oldn2;

            }



            num2 = Math.abs(num2); // Correctly place outside the loop to
finalize the positive value

        } catch (InputMismatchException ime) { // Add a variable for the
caught exception
```

```
            System.err.println("Error: input must be two numbers.");
            System.exit(1);
        } finally {
            stdin.close(); // Close the scanner to avoid resource leak
        }

        // Fix print statement format and variable references
        System.out.println("The GCD of " + input1 + " and " + input2 + " is
" + num2);
    }
}
```

Q5: Consider this simple Java program.

```
1.      class SensorObj{
2.      String name;
3.      Object state; // an attribute of class Object

4.      SensorObj(String sensorName, Object o) {
5.      name=sensorName;
6.      state= o;
7.      }

8.      String getSensorName (){
9.      return name;
10.     }

11.     Object getSensor (){
12.     return state;
13.     }

14.     void showSensorType(){
15.     System.out.println( "The type of the object is
"+state.getClass().getName());
16.     }
17.     }

18.     class SensorDemo {
19.     public static void  main(String args[] ){

20.     SensorObj smokeSensor;

21.     smokeSensor = new SensorObj("Smoke sensor", false) ;

22.     smokeSensor.showSensorType();
23.     boolean v = (Boolean) smokeSensor.getSensor();
24.     System .out.println(smokeSensor.getSensorName()+" : " + v);
```

```
25.     System.out.println();

26.     SensorObj humiditySensor = new SensorObj( "Humidity sensor", "Dry"
);
27.     humiditySensor.showSensorType();
28.     String str = (String) humiditySensor.getSensor( ) ;
29.     System.out.println (humiditySensor.getSensorName()+": "+ str);

30.     humiditySensor=smokeSensor;
31.     str = (String) humiditySensor.getSensor();
```

Answer the following questions:

1. Can the code be compiled? If not which line(s) of the code generate(s) an error? Provide your reasoning.
2. Can the code be executed? If not which line(s) of the code generate(s) an error? Provide your reasoning.
3. Rewrite the code such that:
    a. SensorObj class is declared as a generic class, and
    b. Three instances (having different type) of the generic class are defined in the Main method:
        i. a smoke sensor
        ii. a humidity sensor, and
        iii. a temperature sensor.

For the full mark provide comments for your code.

Answers

1. The code can be compiled, despite the incorrect type assignment on line 30.
2. The code cannot be executed, because of incorrect type casting on line 31.
3. This code is:

```
class SensorGen<T> {
        String name;
        T state; // an attribute of class Object

        SensorGen(String sensorName, T o) {
                name = sensorName;
                state = o;
        }

        String getSensorName() {
                return name;
        }
```

```
        T getSensor() {
                return state;
        }

        void showSensorType() {
                System.out.println("The type of the object is  " +
state.getClass().getName());
        }
}

public class SensorGenDemo {
        public static void main(String args[]) {

                SensorGen<Boolean> smokeSensor = new SensorGen("Smoke
sensor", false);
                SensorGen<String> humiditySensor = new SensorGen("Humidity
sensor", "Dry");
                SensorGen<Double> temperatureSensor = new
SensorGen("Temperature sensor", 20.0);

                smokeSensor.showSensorType();
                boolean v = (Boolean) smokeSensor.getSensor();
                System.out.println(smokeSensor.getSensorName() + " : " +
v);
                System.out.println();

                humiditySensor.showSensorType();
                String str = (String) humiditySensor.getSensor();
                System.out.println(humiditySensor.getSensorName() + ": " +
str);
                System.out.println();

                temperatureSensor.showSensorType();
                double temp = (Double) temperatureSensor.getSensor();
                System.out.println(temperatureSensor.getSensorName() + ": "
+ temp);
        }
}
```