Problem Set 3, Part I

Please start your answer to each problem on a new page, as we have done below!

Problem 1: A `Temperature` class

1-1)

a) type of method: Mutator

b) header: public void convert(String scale)

1-2)

a) type of method: Accessor

b) header: pub boolean colderThan(Temperature otherTemp)

1-3)

a) problems in code: The temperature and scale fields would be inaccessible without a getter or mutator because they are private fields.

b) rewritten version:

```
Temperature t1 = new Temperature(5.0, "celsius");
System.out.println(t1.getTemperature() +" degrees "+ t1.getScale());
t1.setTemperature(t1.getTemperature() * 2); // Make it twice as hot :-)
```

c) toString

```
public String toString() {
      return temperature + " degrees " + scale;
}
```

Problem 2: A class that needs your help

2-1) You get the can not access error because it is a static method which belongs to the class rather than a specific object and therefore can't read private fields from a specific object. To fix it you just turn it from a static method into an instance method by removing the static keyword.

Revise the code found below:

```
public class FlightTicket {
	// Fields to store seat code and seat row
	private String seatCode;
	private int seatRow;

	// Method to check if the seat is a window seat (A or F)
	public boolean isWindowSeat() {
		return seatCode.equals("A") || seatCode.equals("F");

	// add the new methods here
	// Constructor for FlightTicket
	public FlightTicket(String seatCode, int seatRow) {
		this.setSeatRow(seatRow);
		this.setSeatCode(seatCode);
	}

	// Accessor method for seatrow
	public int getSeatRow() {
		return seatRow;
	}

	// Mutator method for seatRow
	public void setSeatRow(int seatRow) {
		if ((seatRow > 0) && (seatRow < 31)) {
			this.seatRow = seatRow;
		}
		else {
			throw new IllegalArgumentException();
		}
	}
```

```
	// Accessor method for seatCode
	public String getSeatCode() {
		return seatCode;
	}


	// Mutator method for seatCode
	public void setSeatCode(String seatCode) {
		if (seatCode.equals(“A”) || seatCode.equals(“B”) ||
		seatCode.equals(“C”) || seatCode.equals(“D”) ||
		seatCode.equals(“E”) || seatCode.equals(“F”)) {
			this.seatCode = seatCode;
		}
		else {
			throw new IllegalArgumentException();
		}
	}
}
```

2-2) Extended FlightTicket class

Problem 3: Static vs. non-static

3-1)

type and name of the variable	**static or non-static?**	**purpose of the variable, and why it needs to be static or non-static**
double height	non-static	stores the height associated with a given BMI object; needs to be non-static so every BMI object will have its own instance of this variable
double weight	non-static	stores the weight associated with a given BMI object; needs to be non-static so every BMI object will have its own instance of this variable
int underweight	static	stores the number of underweight BMI objects calculated throughout the program; needs to be static so it can globally track every BMI object across the program without being associated to one singular object
int normal	static	stores the number of normal BMI objects calculated throughout the program; needs to be static so it can globally track every BMI object across the program without being associated to one singular object
int overweight	static	stores the number of overweight BMI objects calculated throughout the program; needs to be static so it can globally track every BMI object across the program without being associated to one singular object
int obese	static	stores the number of obese BMI objects calculated throughout the program; needs to be static so it can globally track every BMI object across the program without being associated to one singular object
String bmi	non-static	stores the category associated with a given BMI object after calculation; needs to be non-static so every BMI object will have its own instance of this variable

3-2)

a) static or non-static?: non-static
   explanation: it needs to access and mutate object specific fields

b) changes it would need to make:

The method would have to first have to remove 1 from the appropriate static class variable associated with its original BMI in order to maintain an appropriate count. Then it would need to update the height and weight of the objects, then would need to update the bmi string itself with the proper bmi, and finally would need to add 1 to the appropriate static class variable corresponding with the new BMI.

3-3)

a) static or non-static?: static

explanation: It has no need to access or mutate object specific fields

b) example of calling it: BMI.computeWHR(30.0, 38.0);

3-4)

a) static or non-static?: non-static

explanation: It needs to access and mutate object specific variables (weight)

b) example of calling it: b.loseWeight(1000.0);