1. **Choosing fields** For each of the following kinds of values, choose appropriate fields to represent the value (imagine you are trying to implement a class that represents the value). Try to come up with two alternate sets of fields that could represent each kind of value.

   (a) weight

   Solution: There are many possible solutions; two are given below.

   (a) for weights in kilograms:
   ```
   double kilos;
   ```
   (b) for weights in some arbitrary unit:
   ```
   double quantity;
   String units;   // but a Units class would be better than String
   ```

   (b) temperature

   Solution: There are many possible solutions; two are given below.

   (a) for temperatures in degrees Celcius:
   ```
   double degreesC;
   ```
   (b) for temperature in some arbitrary unit:
   ```
   double degrees;
   String units;   // but a Units class would be better than String
   ```

   (c) time of the day

   Solution: There are many possible solutions; two are given below.

   (a) for hours and minutes:
   ```
   int hour;
   int minutes;
   ```
   (b) for minutes after midnight:
   ```
   int minutesFromMidnight;
   ```

   (d) day of the year

   Solution: There are many possible solutions; two are given below.

   (a) for days from January 1:
   ```
   int day;
   ```
   (b) for month and day:
   ```
   int month;  // or String or a Month class
   int dayOfMonth;
   ```

2. **Default constructor**

   (a) Suppose that a `Temperature` is represented as a floating point value in degrees Celcius. Implement a default (no argument) constructor.

Solution:

```
public class Temperature {
  private double degC;

  public Temperature() {
    this.degC = 0.0;    // 0 degrees Celcius
  }
}
```

(b) Suppose that a `TimeOfDay` is represented as an integer hour and an integer minute. Implement a default (no argument) constructor.

Solution:

```
public class TimeOfDay {
  private int hour;     // 24-hour clock: 0 <= hour <= 23
  private int minute;   // 24-hour clock: 0 <= minute <= 59

  public TimeOfDay() {
    this.hour = 0;      // midnight
    this.minute = 0;
  }
}
```

3. **Custom constructor**

   (a) Suppose that a `Temperature` is represented as a floating point value in degrees Celcius. Implement a custom constructor that initializes the temperature given a value in degrees Celcius.

Solution:

```
public class Temperature {
  private double degC;

  public Temperature(double degC) {
    // physics students: should you validate degC?
    this.degC = degC;
  }
}
```

   (b) Suppose that a `TimeOfDay` is represented as an integer hour and an integer minute. Implement a custom constructor that initializes a time given an hour and a minute.

Solution:

```
public class TimeOfDay {
  private int hour;     // 24-hour clock: 0 <= hour <= 23
  private int minute;   // 24-hour clock: 0 <= minute <= 59

  public static final int MIN_HOUR = 0;
  public static final int MAX_HOUR = 23;
  public static final int MIN_MINUTE = 0;
  public static final int MAX_MINUTE = 59;


  public TimeOfDay(int hour, int minute) {
    if (hour < MIN_HOUR || hour > MAX_HOUR) {
      throw new IllegalArgumentException("bad hour");
    }
    if (minute < MIN_MINUTE || minute > MAX_MINUTE) {
      throw new IllegalArgumentException("bad minute");
    }
    this.hour = hour;
    this.minute = minute;
  }
}
```

4. **Copy constructor**

   (a) Suppose that a `Temperature` is represented as a floating point value in degrees Celcius. Implement a copy constructor that initializes the temperature given another `Temperature` reference.

Solution:

```
public class Temperature {
```

```
  private double degC;

  public Temperature(Temperature other) {
    this.degC = other.degC;
  }
}
```

(b) Suppose that a `TimeOfDay` is represented as an integer hour and an integer minute. Implement a copy constructor that initializes a time given another `TimeOfDay` reference.

Solution:

```
public class TimeOfDay {
  private int hour;     // 24-hour clock: 0 <= hour <= 23
  private int minute;   // 24-hour clock: 0 <= minute <= 59

  public static final int MIN_HOUR = 0;
  public static final int MAX_HOUR = 23;
  public static final int MIN_MINUTE = 0;
  public static final int MAX_MINUTE = 59;


  public TimeOfDay(TimeOfDay other) {
    this.hour = other.hour;
    this.minute = other.minute;
  }
}
```

5. **Implement a set method**

   (a) Suppose that a `Temperature` is represented as a floating point value in degrees Celcius. Implement a `set` method that sets the value of a temperature given a value in degrees Celcius.

   Solution:

```
public class Temperature {
  private double degC;

  public void set(double degC) {
    this.degC = degC;
  }
}
```

   (b) Suppose that a `TimeOfDay` is represented as an integer hour and an integer minute. Implement a `set` method that sets the value of a time given an hour and a minute.

   Solution:

```
public class TimeOfDay {
  private int hour;     // 24-hour clock: 0 <= hour <= 23
  private int minute;   // 24-hour clock: 0 <= minute <= 59

  public static final int MIN_HOUR = 0;
  public static final int MAX_HOUR = 23;
  public static final int MIN_MINUTE = 0;
  public static final int MAX_MINUTE = 59;


  public void set(int hour, int minute) {
    if (hour < MIN_HOUR || hour > MAX_HOUR) {
      throw new IllegalArgumentException("bad hour");
    }
    if (minute < MIN_MINUTE || minute > MAX_MINUTE) {
      throw new IllegalArgumentException("bad minute");
    }
    this.hour = hour;
    this.minute = minute;
  }
}
```

6. **toString**

   Implement a `toString` method for `Temperature` and `TimeOfDay`.

   Solution: The implementation of toString will depend on what you decide the string representation of a temperature or time of day should look like. The solutions below illustrate only one possible implementation for each class.

```
public class Temperature {
```

```
  private double degC;

  @Override
  public String toString() {
    return this.degC + " degrees Celcius";
  }
}
```

Solution:

```
public class TimeOfDay {
  private int hour;     // 24-hour clock: 0 <= hour <= 23
  private int minute;   // 24-hour clock: 0 <= minute <= 59

  public static final int MIN_HOUR = 0;
  public static final int MAX_HOUR = 23;
  public static final int MIN_MINUTE = 0;
  public static final int MAX_MINUTE = 59;

  @Override
  public String toString() {
    if (this.minute < 10) {
      // add a zero in front of the single digit minute
      return this.hour + ":0" + this.minute;
    }
    return this.hour + ":" + this.minute;
  }
}
```

7. **equals**

Consider a `Card` class that represents a standard playing card. Every `Card` object has a `Rank` and a `Suit`. Complete the `equals` method for `Card`. You may assume that the `Rank` and `Suit` classes both have an `equals` method.

```
public class Card implements Comparable<Card> {

   private Rank rank;
   private Suit suit;

   /**
    * Compares this playing card to the specified object. The result
    * is <code>true</code> if and only if the argument is a
    * <code>Card</code> with the same rank and suit as this card.
    *
    * @param obj
    *            The object to compare this Card against.
    * @return true if the given object is a
    *         Card equal to this playing card,
    *         false otherwise.
    */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {    // is this Card and obj the same object?
           return true;
        }
        if (obj == null) {    // equals(null) is always false
            return false;
        }
        if (getClass() != obj.getClass()) {    // is obj a Card?
            return false;
        }
        Card other = (Card) obj;  // cast obj to a Card

        // only now can you compare the ranks and suits
        return this.getRank().equals(other.getRank()) &&
               this.getSuit().equals(other.getSuit());
        }
}
```