Quiz 1 (March 23, 2018)

Your name: _____

Your Kerberos username: _____

You have 50 minutes to complete this quiz. It contains 8 pages (including this page) for a total of 100 points.

The quiz is closed-book and closed-notes, but you are allowed one two-sided page of notes.

Please check your copy to make sure that it is complete before you start. Turn in all pages, together, when you finish. Before you begin, write your Kerberos username on the top of every page.

Please write neatly. **No credit will be given if we cannot read what you write.**

For questions which require you to choose your answer(s) from a list, do so clearly and unambiguously by circling the letter(s) or entire answer(s). Do not use check marks, underlines, or other annotations – they will not be graded.

Good luck!

Problem	Points
1: Code review	20
2: Specs	22
3: Equality	22
4: Reps	36
Total	100

In sciences like physics and chemistry, *dimensional analysis* is often used to check calculations for errors.

Dimensional analysis associates a number with a unit of measure, called its *dimension*. Examples of dimensioned numbers include:

- a length of 10 meters (m)
- a time of 0.8 seconds (s)
- a relative velocity of -20 meters per second (m/s, or ms^{-1})
- an acceleration of 9.8 meters per second per second (m/s^2)
- an area of 100 square meters (m^2)
- a volume change of -0.001 cubic meters (m^3)

Less familiar but still valid examples of dimensioned numbers include:

- 99.9 seconds per meter (s/m)
- 2 meter-seconds ($m \cdot s$)

A number's dimension can be empty, or *dimensionless*, like π.

Two rules of dimensional analysis are:

- Numbers with different dimensions should not be added, subtracted, or compared. For example, it makes no sense to compare a length in meters with a time in seconds, or to add them together.
- The ratio (or product) of two dimensioned numbers produces a number whose dimension is the ratio (or product) of the two dimensions. For example, a velocity in m/s multiplied by a time in seconds produces a length in meters.

The problems in this quiz refer to the code for `DimDouble` on page 7, which you may detach.

The back side of that page has abbreviated specs for Java's `List` and `Map` that may be useful if you need them.

The last problem of this quiz takes longer than the earlier problems. Use your time well.

Problem 1 (Code review) (**20 points**).
The code for **`DimDouble`** is buggy. For each of these code review comments, circle whether you AGREE or DISAGREE with the comment, and explain why in one sentence.

(a) Line 12: remove `final` from `result`, because it prevents changing `result` in lines 13–14.

AGREE / DISAGREE because:

(b) Lines 13–14: this code creates aliasing between the reps of different `DimDouble` objects.

AGREE / DISAGREE because:

(c) Line 28: this code violates the spec of `append`.

AGREE / DISAGREE because:

(d) Lines 29–30: this code violates the spec of `append`.

AGREE / DISAGREE because:

Problem 2 (Specs) (**22 points**).
Consider this spec for a method that uses `DimDouble`:

```
/**
 * Compute the radius of a hypersphere, the set of points equidistant
 * from a center point in N-dimensional space.  For example, a 3-sphere is
 * a conventional sphere in 3D, and a 2-sphere is a circle in 2D.
 *
 * @param volume must be > 0 with units in meters^N for N > 0
 * @return the radius of an N-sphere with the given volume, in meters
 */
public DimDouble radiusOfHyperSphere(DimDouble volume)
```

Here is an example of a correct call to this method:

```
radiusOfHyperSphere(new DimDouble(2, METERS))
```

For each of the following *buggy* calls to the method, circle whether the **spec guarantees that** the bug is caught by a static error, by a dynamic error, or no guarantee that the bug is caught.

(a) `radiusOfHyperSphere(21.3, METERS)`

STATIC ERROR DYNAMIC ERROR NO GUARANTEE

(b) `radiusOfHyperSphere(new DimDouble(-5, METERS))`

STATIC ERROR DYNAMIC ERROR NO GUARANTEE

(c) `radiusOfHyperSphere(new DimDouble(7, SECONDS))`

STATIC ERROR DYNAMIC ERROR NO GUARANTEE

(d) `Math.PI * radiusOfHyperSphere(new DimDouble(0.5, METERS))`

STATIC ERROR DYNAMIC ERROR NO GUARANTEE

Louis Reasoner proposes several partitions for choosing test cases for `radiusOfHyperSphere()` based on the `volume` parameter and the radius returned. Alyssa likes all of his partitions *except* this one:

```
volume is dimensionless, in meters, or in seconds
```

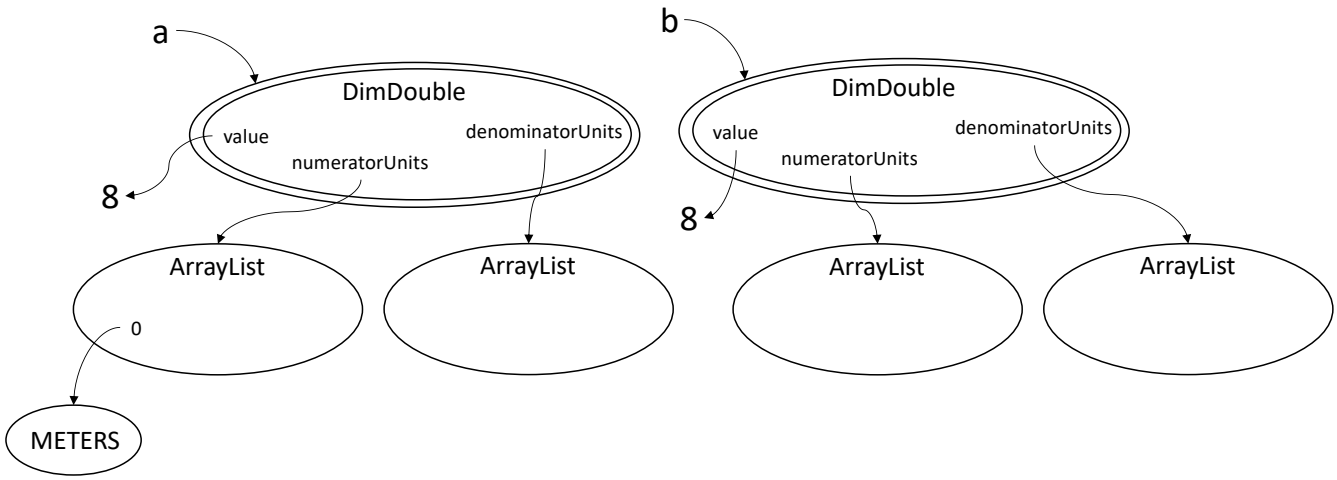
Give two (substantially different) reasons why this proposal is not a good partition. Each reason should be one sentence in its own box below.

(e)

(f)

Problem 3 (Equality) (**22 points**).
Ben Bitdiddle points out that `DimDouble` is missing not only specs, but also an abstraction function and rep invariant. The implementations for `multiply()` and `divide()` give some clues about what reps are intended to be legal and how the rep is intended to be interpreted.

(a) Complete the snapshot diagram below to form two different-looking rep values `a` and `b` that should have the same abstract value.



(b) Write the abstract value that your `a` and `b` represent.

(c) Suppose `DimDouble` uses the following helper function as part of implementing its equality operation.

```
private boolean sameAs(DimDouble that) {
    return this.value == that.value
        && this.numeratorUnits  .containsAll(that.numeratorUnits  )
        && this.denominatorUnits.containsAll(that.denominatorUnits);
}
```

Which properties of an equivalence relation does this function satisfy, and which does it violate? Fill in each blank with a name, and circle YES if it's satisfied and NO if not.

_____ YES NO

_____ YES NO

_____ YES NO

Problem 4 (Reps) (**36 points**).
Alyssa Hacker suggests an alternative rep for `DimDouble`:

```
private double value;
private Map<Unit, Integer> units = new HashMap<>();
```

Write an abstraction function, rep invariant, and two operations using this new rep. Read this whole page before making design decisions. To simplify, you can assume for this entire page that the only base units are `METERS` and `SECONDS`.

(a) Abstraction function

(b) Rep invariant (implicit 6.031 rep invariant can be omitted)

Complete the two operations below using this new rep. Note that part of the code has been filled in already.

(c)
```
/** make a dimensioned number equal to ‘value‘ with units ‘base‘ */
public DimDouble(double value, Unit base) {
    this.value = value;
```

```
    checkRep();
}
```

(d)
```
/** @return the product of ‘this‘ and ‘that‘ */
public DimDouble multiply(DimDouble that) {
    DimDouble result = new DimDouble(this.value * that.value);
```

```
    result.checkRep();
    return result;
}
```

You may detach this page. Write your username at the top, and hand in all pages when you leave.

```
public enum Unit { METERS, SECONDS }

/** Immutable dimensioned number. */
public class DimDouble {
    private double value;
    private List<Unit> numeratorUnits = new ArrayList<>();
    private List<Unit> denominatorUnits = new ArrayList<>();

    public DimDouble(double value) {
        this.value = value;
    }

    public DimDouble(double value, Unit base) {
        this.value = value;
        this.numeratorUnits.add(base);
    }

    public DimDouble add(DimDouble that) {
        final DimDouble result = new DimDouble(this.value + that.value);
        result.numeratorUnits = this.numeratorUnits;
        result.denominatorUnits = this.denominatorUnits;
        return result;
    }

    public DimDouble multiply(DimDouble that) {
        final DimDouble result = new DimDouble(this.value * that.value);
        result.numeratorUnits   = append(this.numeratorUnits,   that.numeratorUnits);
        result.denominatorUnits = append(this.denominatorUnits, that.denominatorUnits);
        return result;
    }

    public DimDouble divide(DimDouble that) {
        final DimDouble result = new DimDouble(this.value / that.value);
        result.numeratorUnits   = append(this.numeratorUnits,   that.denominatorUnits);
        result.denominatorUnits = append(this.denominatorUnits, that.numeratorUnits);
        return result;
    }

    /** @return a list of the elements of list1 followed by the elements of list2 */
    private static List<Unit> append(List<Unit> list1, List<Unit> list2) {
        assert list1 != list2;
        List<Unit> result = list1;
        result.addAll(list2);
        return result;
    }

    // ... more operations
}
```

```
interface List<E> {
  /** Modifies this list by appending e to the end of it. */
  public void add(E e);

  /** Modifies this list by appending the elements of l to the end, in order. */
  public void addAll(List<E> l);

  /** @return the number of elements in this list. */
  public int size();

  /** @return true iff e is an element of this list. */
  public boolean contains(E e);

  /** @return true iff every element of l is an element of this list. */
  public boolean containsAll(List<E> l);

  /** @return the element at position index in this list
      @throws IndexOutOfBoundsException if index is out of range for this list */
  public E get(int index);

  /** Modifies this list by removing the first occurrence of e from this list, if any. */
  public void remove(E e);

  /** Modifies this list by removing all elements in this list that are also in l.*/
  public void removeAll(List<E> l);

  // ... more operations
}

interface Map<K,V> {
  /** Modifies this map by mapping key to value, replacing any previous mapping for key. */
  public void put(K key, V value);

  /** @return the value to which key is mapped, or null if key not mapped. */
  public V get(K key);

  /** @return value to which key is mapped, or defaultValue if key not mapped. */
  public V getOrDefault(K key, V defaultValue);

  /** Modifies this map by removing the value associated with key, if any. */
  public void remove(K key);

  /** @return the number of key-value mappings in this map. */
  public int size();

  /** @return a Set view of the keys contained in this map. */
  public Set<K> keySet();

  // ... more operations
}
```