CS 61B Spring 2017 Guerrilla Section 3 Worksheet

25 February 2017

Directions: In groups of 4-5, work on the following exercises. Do not proceed to the next exercise until everyone in your group has the answer and *understands why the answer is what it is*. Of course, a topic appearing on this worksheet does not imply that the topic will appear on the midterm, nor does a topic not appearing on this worksheet imply that the topic will not appear on the midterm.

1 Probably Equal?

(a) Warm-up!

- What is the difference between == and `.equals`?

- Would it make sense for the scenarios below to occur? Explain why or why not.

i `A.equals(B)` but `A != B`

ii `A == B` but `!A.equals(B)`

(b) Write a `probablyEquals` method that takes in two objects and returns true if one or more of the following are true:

- The two objects are equal (`.equals`) to each other.
- The two objects are equal (`==`) to each other.
- The two objects have the same `.toString()` representation.

Otherwise, probablyEquals returns false.
Your method should never crash given **any** input.
You may assume that for any object instances x and y, `x.equals(y)` will return the same value as `y.equals(x)`.
You may also assume that every object has a unique non-random toString representation.

Note: `.equals(Object o)` and `.toString()` are methods that every object subclass inherits from the `Object` class.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

2 Xzibit's ADTs

Consider the following abstract data type definitions:

```
List <E> {
    void insert(E item, int position);
    E remove(int position);
    E get(int position);
    int size();
}

Set <E> {
    void add(E item);
    void remove(E item);
    boolean contains(E item);
    Iterator<E> list();
}

Stack <E> {
    void push(E item);
    E pop();
    boolean isEmpty();
}

Queue <E> {
    void enqueue(E item);
    E dequeue();
    boolean isEmpty();
}

Map<K, V> {
    put(K key, V value);
    remove(K key);
    V get(K key);
    Iterator<K> keys();
}
```

For the following questions, you may assume the above data types have been implemented as classes.

(a) Write an extension of the `Set` class, called `IntegerSet`, with the following methods:

- `void add(Integer item)`: adds an integer to the set
- `boolean contains(Integer item)`:checks item for membership in the set
- `Iterator<Integer> list()`:returns an iterator over the elements of the IntegerSet
- `Integer max()`:returns the maximum value in the IntegerSet, or null if the set is empty

(b) Consider the following PriorityQueue interface:

```
public interface PriorityQueue<E> {
        public void enqueue(E item, int priority);
        public E dequeue();
        public E peek();
}
```

Describe how you would implement this abstract data type using any combination of the above ADT definitions, including `IntegerSet`.

3 Expandable Set

Using inheritance, define a class `TrackedQueue` that behaves like `Queue` except for an extra method, `maxSizeSoFar()` which returns an integer corresponding to the maximum number of elements in this queue since it was constructed. Assume that the `Queue` class is a non-abstract class with the following methods defined:

- `void enqueue(Object obj)`
- `int size()`
- `Object dequeue()`

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

4 Delegation vs. Extension

Consider the following class.

```
interface MyStack<T> {
        void push(T item);
        T pop();
        int size();
}
```

(a) Implement `ExtensionStack` which implements `MyStack` using extension, without using the `new` keyword. (Hint: `ExtensionStack` can extend `LinkedList`).

```
public class ExtensionStack<T>                                          {









}
```

(b) Implement `DelegationStack` which implements `MyStack` using delegation.

```
public class DelegationStack<T>                                 {
        LinkedList<T> data =
        int size() =









}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

5 FunkySets & Promotion

Cross out the lines that would cause compilation errors for each of the classes.
Write out the values that will be printed where indicated in the code's comments.

```
1  import java.util.HashSet;
2  public class FunkySet {
3      public static void main(String[] args){
4          HashSet<int> set = new HashSet<int>();
5          HashSet<Integer> set = new HashSet<int>();
6          HashSet<int> set = new HashSet<Integer>();
7          HashSet<Integer> set = new HashSet<Integer>();
8          int x = 3;
9          set.add(x);
10         set.add(4);
11         Integer y = 5;
12         set.add(y);
13         System.out.println(set.toString());
14         if (set.contains(x)){
15             set.remove(x);
16         }
17         if (set.contains(4)){
18             int z = 4;
19             set.remove(z);
20         }
21         if (set.contains(y)){
22             set.remove(y);
23         }
24         System.out.println(set.toString());
25     }
26 }
27 public class FunkySetTwo {
28     public static void main(String[] args){
29         int [][] x = new int[2][3];
30         Integer [][] y = new Integer[2][3];
31         Integer [][] z = new int[2][3];
32         Integer[] arrayOne = {1,2,3};
33         int[] arrayTwo = {4,5,6};
34
35         x[0] = arrayOne;
36         x[1] = arrayTwo;
37
38         y[0] = arrayOne;
39         y[1] = arrayTwo;
40
41         z[0] = arrayOne;
42         z[1] = arrayTwo;
43     }
44 }
```

```
public class Promotion {
  public static void doublePrinter(double num){
    System.out.println(num);
  }
  public static void longPrinter(long num){
    System.out.println(num);
  }
  public static void intPrinter(int num){
    System.out.println(num);
  }
  public static void intPrinterTwo(Integer num){
    System.out.println(num);
  }
  public static void shortPrinter(short num){
    System.out.println(num);
  }
  public static void main(String[] args){
    int x = 45;
    longPrinter(x);
    doublePrinter(x);
    intPrinter((int)x);
    intPrinterTwo(x);
    shortPrinter(x);
    shortPrinter((short) x);
  }
}
```

6 Iterators

Print ALL bark combinations of dogs from two dog lists in form of 'Woof DOG1! Woof DOG2!'.
The dogs in `dogs1` should bark first. For 3 dogs in `dogs1` and 5 dogs in `dogs2`, there must be 15 bark combinations printed.

```
public class Dog {
    String name = ...;
    public void bark() {
        System.out.print("Woof " + name + "!");
    }
}

public static void barkCombinations(Iterable<Dog> dogs1, Iterable<Dog> dogs2) {
  // YOUR CODE HERE
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

7 Iterator or Iterable?

Implement the `Filter` class such that its `main` method correctly prints out the even numbers in the given collection. (Should print out 0 20 14 50 66 all on newlines)

```
public interface FilterCondition<T> {
    /** Evaluates if the given item passes a certain condition (ie, is even, a prime
        number, is icky, etc.) */
    public boolean eval(T item);
}
public class EvenCondition implements FilterCondition<Integer> {
    public boolean eval(Integer i) {
        return i % 2 == 0;
    }
}
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
public class Filter implements Iterable<Integer> {



    public Filter(Iterable<Integer> thingamajig, FilterCondition<Integer> cond) {


    }

    public Iterator<Integer> iterator() {

    }

    private class FilterIterator implements Iterator<Integer> {
```

```
    }

    public static void main(String[] args) {
        List<Integer> arr = Arrays.asList(new Integer[]{0, 11, 20, 13, 14, 50, 66});
        for (int i : new Filter(arr, new EvenCondition())) {
            System.out.println(i);
        }
    }
}
```

8 Except Me for Who I Am

Consider the following:

```
1  public class IntList {
2      private int head;
3      private IntList tail;
4
5      /* Returns the index of an element in the list */
6      public int getIndex(int item) {
7          int index = 0;
8          IntList temp = this;
9          while(temp.head != item) {
10             temp = temp.tail;
11             index++;
12         }
13         return index;
14     }
15
16     public int getIndexThrowException(int item) throws IllegalArgumentException {
17         // YOUR CODE HERE
18     }
19
20     public int getIndexDefaultNegative(int item) {
21         // YOUR CODE HERE
22     }
23 }
```

(a) What happens when you call `getIndex(int item)` on an element that is not in the list?

(b) Write `getIndexThrowException`, which attempts to get the index of an item, but throws an `IllegalArgumentException` with a useful message if no such item exists in the list. Do not use if statements, while loops, for loops, or recursion. (Hint: you can use `get(int item)`)

(c) Write `getIndexDefaultNegative`, which attempts to get the index of an item, but returns -1 if no such item exists in the list. Again, do not use if statements, while loops, for loops, or recursion.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

9 Generic Brand Generics

For each of the modifications of maxKey from lecture (found on the next page), identify which lines fail to compile in the main method of `MapHelperUser`:

```
public class MapHelper {
    /* maxKey belongs here */
}
```

```
class Dog {
}

class Cat implements OurComparable {
    public int compareTo(Object o) {
        /* CODE NOT SHOWN */
    }
}

class Parrot implements Comparable<Parrot> {
    public int compareTo(Parrot o) {
        /* CODE NOT SHOWN */
    }
}

class Penguin<T> implements Comparable<Penguin<T>> {
    public int compareTo(Penguin<T> o) {
        /* CODE NOT SHOWN */
    }
}

class Sloth implements Comparable<T> {
    public int compareTo(T o) {
        /* CODE NOT SHOWN */
    }
}
```

```
class MapHelperUser {
    public static void main(String[] args) {
        Map<Dog, Integer> dogMap = new ArrayMap<>();
        Map<Integer, Dog> mapDog = new TreeMap<>();
        Map<Cat, String> catMap = new ArrayMap<>();
        Map<Parrot, Boolean> parrotMap = new ArrayMap<>();
        Map<Penguin<Float>, Long> penguinMap = new HashMap<>();
        Map<Sloth, Double> slothMap = new ArrayMap<>();

        // Assume maps are filled

        Dog topDog = MapHelper.maxKey(dogMap);
        Integer numberOne = MapHelper.maxKey(mapDog);
        Cat coolestCat = MapHelper.maxKey(catMap);
        Parrot partyParrot = MapHelper.maxKey(parrotMap);
        Penguin<Float> supremePenguin = MapHelper.maxKey(penguinMap);
        Sloth slowSloth = MapHelper.maxKey(slothMap);
    }
}
```

Greater Than

```
public static <K, V> K maxKey(Map61B<K, V> map) {
    List<K> keyList = map.keys();
    K largest = keylist.get(0);
    for (K k : keyList) {
        if (k > largest) {
            largest = k;
        }
    }
    return largest;
}
```

compareTo (What's wrong with just using <K, V>?)

```
public static <K, V> K maxKey(Map61B<K, V> map) {
    List<K> keyList = map.keys();
    K largest = keylist.get(0);
    for (K k : keyList) {
        if (k.compareTo(largest) > 0) {
            largest = k;
        }
    }
    return largest;
}
```

OurComparable

```
// Assume same body as above
public static <K extends OurComparable, V> K maxKey(Map61B<K, V> map)
```

Comparable<K>

```
// Assume same body as above
public static <K extends Comparable<K>, V> K maxKey(Map61B<K, V> map)
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!