# CS61B Spring 2015 Guerrilla Section 1 Worksheet SOLUTIONS

Leo Colobong, Nick Fong, Jasmine Giang, Andrew Huang, Yujie Huang, Nick Rose, Khalid Shakur, Jason Won

15 February 2015

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. Code on this worksheet can be downloaded from Nick's repo[1].

## 1 Potpourri

(a) Find the bugs in the code below so that main prints out "Doge coin:100.45"

Knapsack.java

```
class Knapsack {
    public String thing;
    public String amount;

    public Knapsack(String str, double amount) {
        String thing = str;
        amount = amount;
    }

    public Knapsack(String str) {
        Knapsack(str, 100.45);
    }

    public static void main(String     args){
        Knapsack sack = new Knapsack("Doge coin");
        System.out.println(thing + " : " + amount);
    }
}
```

solutions/KnapsackSolution.java

```
class Knapsack {
    public String thing;
    public double amount;                          //Changed

    public Knapsack(String str, double amount) {
        this.thing = str;                          //Changed
        this.amount = amount;                      //Changed
    }
```

[1] https://github.com/Fong-/CS61B-Guerrilla-Section-1

```
9
10      public Knapsack(String str) {
11          this(str, 100.45);                          //Changed
12      }
13
14      public static void main(String     args){
15          Knapsack sack = new Knapsack(  D o g e  c o i n  );
16          System.out.println(sack.thing +     :     + sack.amount); //Changed
17      }
18  }
```

(b) Type casting of a variable (e.g. `(String) x`) permanently changes the variable's

(i) static type

(ii) dynamic type

(iii) (i) and (ii)

(iv) None of the above

(c) Write the 8-bit two's complement binary representation of the following decimal numbers.

(i) 7 00000111

(ii) 64 01000000

(iii) -64 11000000

(iv) -61 11000011

(d) Convert the following 8-bit binary numbers to decimal.

(i) 00101000 40

(ii) 01111111 127

(iii) 01010101 85

(iv) 10000011 -125

# STOP!

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

## 2 Bit Manipulation

**From Fall 2014 Discussion 6**

(a) Assuming $x_1, x_2, x_3, \ldots x_n$ are integers and that $\oplus$ is the logical symbol denoting `XOR`, what is $(x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_n) \oplus (x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_n)$?
**Solution:** $i \oplus i$ is always 0

(b) Write an expression to check whether a 32-bit integer is less than 0 using only bit operations and `==`.
**Possible Solutions:**

```
(x>>>31) == 1
(x & 0x7fffffff) != x
(x & 0x80000000) != 0
(x << 1 >>> 1) != x
```

(c) What does the following code block do?

mysteryBit.java

```
1 public static int mysteryBit(int n) {
2     return n & (n - 1);
3 }
```

**Solution:** Return $n$ with the rightmost 1 bit set to 0

(d) Write a program to count the number of 1's in an integer represented in base 2.

solutions/countBitsSolution.java

```
1 public static int countBits(int n) {
2     int count = 0;
3     while (n != 0) {
4         n &= (n-1);
5         count++;
6     }
7     return count;
8 }
```

# STOP!

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

# 3 IntLists

For reference, here's a snippet of the `IntList` code as was shown in lecture:

IntList.java

```java
/** Defines a recursive list of integers.
 *  @author Josh Hug
 */

public class IntList {
    public int head;
    public IntList tail;

    public IntList(int h, IntList t) {
        head = h;
        tail = t;
    }

    /** Returns the size of the IntList */
    public int size() {
        if (this.tail == null)
            return 1;
        int personInFrontOfMeSize = this.tail.size();
        return personInFrontOfMeSize + 1;
    }
```

(a) **EvenOdd – Summer 2014 Final**

Write an `evenOdd` method that destructively sets the passed in `IntList` to contain every other `IntList` node of the original `IntList`, starting with the first node. Your method must also return an `IntList` that contains every other `IntList` node of the original `IntList`, starting with the second node. Your method should work destructively and should not create any new `IntList` objects. If an `IntList` contains zero elements or only one element, a call to `evenOdd` should return `null`.

Example: If an `IntList` initially contains the elements [5, 2, 3, 1, 4], then a call to `evenOdd` should return an `IntList` with the elements [2, 1], and after the call, the original `IntList` should contain the elements [5, 3, 4]

solutions/evenOddSolution2.java

```java
public class IntList {
  public int head;
  public IntList tail;

  public IntList(int h, IntList t) {
    head = h;
    tail = t;
  }

  public int size() {
    if (this.tail == null) return 1;
    return tail.size() + 1;
  }

  public static IntList evenOdd(IntList L) {
    if (L == null || L.tail == null) {
      return null;
    }
    IntList toReturn = L.tail;
    IntList p = L.tail;
```

```
21      while (L != null && p != null) {
22        L.tail = p.tail;
23        L = L.tail;
24        if (L != null) {
25          p.tail = L.tail;
26          p = L.tail;
27        }
28      }
29      return toReturn;
30    }
31  }
```

# STOP!

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

(b) Write triangularize, a method that takes in an array of `IntLists` $R$ and a single `IntList` $L$, and breaks $L$ into smaller `IntLists`, storing them into $R$. The `IntList` at index $k$ of $R$ has at most $k+1$ elements of $L$, in order. Thus concatenating all of the `IntLists` in $R$ together in order would give us $L$ back. Assume $R$ is big enough to do this.

For example, if the original L contains [ 1, 2, 3, 4, 5, 6, 7 ], and R has 6 elements, then on return R contains [ [1], [2,3], [4,5,6], [7], [], []]. If R had only 2 elements, then on return it would contain [[1], [2,3]]. Triangularize may destroy the original contents of the `IntList` objects in L, but does not create any new `IntList` objects.

solutions/triangularize–solution.java

```
1  static void triangularize(IntList[] R, IntList L) {
2      int i, k; // i: index into R, k: number of items in R[k]
3      i = 0; k = 0;
4      while (i < R.length) {
5          if (k == 0) {
6              R[i] = L;
7          }
8          if (L == null) {
9              i += 1;
10             k = 0;
11         }
12         else if ( k == i) {
13             IntList next = L.tail;
14             L.tail = null;
15             L = next;
16             i += 1;
17             k = 0;
18         }
19         else {
20             L = L.tail;
21             k += 1;
22         }
23     }
24 }
```

# STOP!

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

# 4 OOP

(a) **Abstract Class – Fall 2007 MT1**
For the code below, answer the following questions:

(i) Does this code compile? If not, what's the compile-time error?

(ii) Does this code run? If not, what's the run-time error?

(iii) If the code does not compile/run, what is the minimum change needed to print "Berkeley!"?

abstraction.java

```
1  abstract class abstraction {
2      abstract void foo();
3  }
4
5  class bar {
6      void foo() {
7              System.out.println("Berkeley!");
8      }
9  }
10
11 public class abstractexample {
12     public static void callfoo(abstraction widget) {
13         widget.foo();
14     }
15
16     public static void main (String[] args) {
17             Object thebar = new bar();
18             callfoo((abstraction)thebar);
19     }
20 }
```

**Solutions:**

(i) Code compiles

(ii) The code does not run. ClassCastException – bar can't be casted to abstraction because an object of class bar is not a subclass of abstraction, despite having the same methods

(iii) Change line 5 to read `class bar extends abstraction {`

STOP!

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

(b) **Static vs. Dynamic**

In the code below, cross out any lines that cause errors and state whether the error is a compile-time or run-time error. Also write out the output of each print statement. You may find the official Object class documentation helpful here.

Vertebrate.java

```
public class Vertebrate {
    public int numLegs = 4;
    public void eat() {
        System.out.println("Generic Vertebrate is eating");
    }
}
```

Bird.java

```
public class Bird extends Vertebrate {
    public int numLegs = 2;
    public void eat() {
        System.out.println("The bird is eating.");
    }
}
```

Run.java

```
public class Run {
    public static void main(String[] args) {
        Vertebrate a = new Vertebrate();
        Bird b = new Bird();
        Vertebrate thebird = b;

        // What will java print?
        System.out.println(a.numLegs);
        System.out.println(b.numLegs);
        System.out.println(thebird.numLegs);
        System.out.println(((Vertebrate) b).numLegs);
        System.out.println(((Bird) thebird).numLegs);

        // What will java do?
        a.eat();
        b.eat();
        thebird.eat();
        ((Vertebrate) b).eat();
        ((Bird) thebird).eat();

        double d = ((Object) "string").length();
    }
}
```

solutions/VertebrateSolutions.txt

```
[s277-12:tmp][Fri Feb 13 17:47:27]$ java Run
4
2
4
4
2
Generic Vertebrate is eating
The bird is eating.
```

```
9  The bird is eating.
10 The bird is eating.
11 The bird is eating.
12 [s277-12:tmp][Fri Feb 13 17:47:30]$
13
14 Line 21 breaks compiling.
```

# STOP!

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

(c) **Pointers**

Quick terminology reminder: Java variables are simple containers that can hold either primitive values or references to Objects. The contents of the container are its value.

Which (if any) of the following method calls has an effect on the value of $x$ or $y$? Which of these method calls has an effect that we could observe using a print statement on the next line after the call?

ObjectPractice.java

```
1  public class ObjectPractice {
2      static void copy1(int a, int b) {
3         a = b;
4      }
5
6      static void copy2(String a, String b) {
7         a = b;
8      }
9
10     static void copy3(String[] a, String[] b) {
11        a[0] = b[0];
12     }
13
14     public static void main(String[] args) {
15         /*
16          * What's the value of x and y after executing each call to copy?
17          * Is it possible to print evidence of running each call?
18          */
19         int x = 5;
20         int y = 10;
21         copy1(x, y)
22
23         String x = "melted";
24         String y = "cheese";
25         copy2(x, y);
26
27         String[] x = new String[] {"friendly", "pickly"};
28         String[] y = new String[] {"bodacious", "horse"};
29         copy3(x, y);
30     }
31 }
```

**Solution:** `copy1` and `copy2` do not affect the values stored in $x$ and $y$. `copy3` mutates $x$ such that it contains {`"bodacious"`, `"pickly"`}.

STOP!

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

(d) **Inheritance – Fall 2014 MT1**

Fill in the blanks and cross out and/or rewrite lines of code in the `Animal` and `Dog` classes so that `Foo.java` compiles and prints out the following lines:

AnimalOutput.txt

```
1
2
3
Superdog
Superdog
bark 3
4
```

Animal.java

```
package zoo;
public class Animal {
    int noise;
    private String name; //Do not modify

    public Animal(String name) {
        name = name;
    }

    public void makeNoise() {
        ________________________________________;
        System.out.println(_____________________);
    }

    public void sayName() {
        System.out.println(name);
    }
    ________________________________________________;
}
```

Dog.java

```
package housepets;
import zoo.Animal;
public class Dog {
    public Dog() {
        ________________________________________;
    }

    public void makeNoise(String sound) {
        System.out.println(sound + " " + _____________________;
    }
}
```

Foo.java

```
import zoo.Animal;
import housepets.Dog;
public class Foo {
    public static void main(String[] args) {
        Animal a = new Dog();
        Animal b = new Dog();
        Animal c = new Dog();
```

```
        a.makeNoise();
        b.makeNoise();
        a.makeNoise();
        c.sayName();
        a.sayName();
        a.makeNoise("bark");
        c.makeNoise();
    }
}
```

solutions/animalSolution.java

```
package zoo;

public class Animal {

    //int noise;
    public static int noise;                //Changed
    private String name; //Do not modify

    public Animal(String name) {            //Changed
        this.name = name;                   //Changed
    /*
     * Alternatively:
     * public Animal(String newName) {
     *     name = newName;
     * }
     */

    public void makeNoise() {
    noise ++;
    System.out.println(noise);
    }

    public void sayName() {
    System.out.println(name);
    }
    public void makeNoise(String sound){}
}


package housepets;
import zoo.Animal;

//public class Dog {
public class Dog extends Animal {           //Changed

    public Dog() {
    super("Superdog");
    }

    public void makeNoise(String sound) {
        System.out.println(sound + " " + noise);
    }
}
```