Note: This is a preliminary version of the worksheet. It it almost complete.

CSM Berkeley 61B, Spring 2015: Week 4 Solutions

1. Bit Manipulation

File: BitManips.java

```
public class BitManips {
```

1a. Rotate a 32-bit integer left by `k` bits. Assume that `k` is less than 32.

```
    int rotateLeft(int x, int k) {
        return (x << k) | (x >> (32 - k));
    }
```

1b. Check if an integer is a multiple of 4 using only the `&` operator and equality checks.

```
    int isMultipleOfFour(int x) {
        // Think: return !(x & 0b11);
        return !(x & 3);
    }
```

1c. Check if an integer is odd using only bit shifting and equality checks.

Assume that you do not know the number of bits in your number.

```
    int isOdd(int x) {
        return x != ((x >> 1) << 1);
    }
```

1d. Write a one-line expression equivalent to `x * 35` without using `*`, `/`, or `%`.

```
    int times35(int x) {
        // 35 = 32 + 2 + 1 = 2^5 + 2^1 + 2^0
        return (x << 5) + (x << 2) + x;
    }
}
```

1e. What does `n & (n - 1) == 0` test? (Fall 2013 Final Exam)

This checks if `n` is a power of 2.

Why? For anything but a power of 2 minus 1, the most significant bit will stay, and so the result will be nonzero.

2. Lists

2a. SLists

Write a method that, given an SList, an int j, and an int k, return an SList with elements k, k+j, k+2*j, ... **Do not change the original list.**

File: Slist.java

```
public class SList {
    private Node head;
    public SList(Node head) {
        this.head = head;
    }

    public static SList multiples(SList list, int j, int k) {
        SList newList = new SList(null);
        Node oldNode = list.head;

        // Get the kth element
        for (int i = 0; i < k; i++) {
            if (oldNode == null) return newList;
            oldNode = oldNode.next;
        }
        newList.head = new Node(oldNode.item);
        Node newNode = newList.head;
        oldNode = oldNode.next;

        // Keep going through the list and add every j
        for (int i = 1; oldNode != null; i++) {
            if ((i % j) == 0) {
                newNode.next = new Node(oldNode.item);
                newNode = newNode.next;
            }
            oldNode = oldNode.next;
        }

        return newList;
    }

    public String toString() {
        String result = "";
        for (Node cur = head; cur != null; cur = cur.next)
            result += cur.item.toString() + " ";
        return result;
    }

    private static Node n(Object item, Node next) {
        return new Node(item, next);
    }
    private static Node n(Object item) {
        return new Node(item);
    }
```

```
    public static void main(String[] args) {
        SList l = new SList(n(0, n(1, n(2, n(3, n(4, n(5, n(6)))))));
        System.out.println(l);
        System.out.println(multiples(l, 2, 0));
        System.out.println(multiples(l, 2, 1));
        System.out.println(multiples(l, 3, 2));
    }
}

class Node {
    Object item; Node next;
    Node(Object item, Node next) {
        this.item = item; this.next = next;
    }
    Node(Object item) {
        this(item, null);
    }
}
```

2b. Arrays

1. [2 points] Assume that a Point's toString method returns a string containing that Points's coordinates (so that System.out.println(x) prints "(4, 5)" if x is new Point(4, 5) and "null" if x is null). What is the output of the following (valid) program?

```
import java.awt.Point;
public class Foo {
    public static void bar (Point[] arr, Point p) {
        arr[1] = p;
        arr[2] = arr[1];
        p.x = 1;
        p = new Point(2,2);
        p.y = 3;
        arr[3] = p;
    }
    public static void main(String[] args){
        Point[] points = new Point[4];
        Point p = new Point(0,0);
        bar(points, p);
        System.out.println(p);
        for (int i = 0; i < points.length; i += 1) {
            System.out.println(points[i]);
        }
    }
}
```

3. Static and dynamic types review

```
List l;
if (use_linked_list) {
    l = new LinkedList();
} else {
    l = new ArrayList();
}
```

static types = the **declared** type = checked at **compile time**
We don't need to run the code to know that `l` is a `List`.

dynamic type = the **actual** type = checked at **run time**
When we run the code, depending on the situation, `l` might either be a `LinkedList` or `ArrayList`.

```
// What would Java do?
Collection c;
if (use_set) {
    c = new HashSet();
} else {
    c = new ArrayList();
}

// Example 1: works!
c.isEmpty();  // works because Collection.isEmpty() exists
c.size();     // works because Collection.size() exists

// Example 2: compile time error
c.sort();  // compile-time error: Collection.sort() doesn't exist
c.get(0);  // compile-time error: Collection.get(int) doesn't exist
```

Static types are like guarantees or agreements. The declaration `Collection c` means that `c` is guaranteed to have `Collection`'s methods, including `isEmpty()` and `size()`. Even though `ArrayList` has some additional methods like `sort()` and `get(int)`, there was no agreement that `c` would be an `ArrayList`, so you can't use these methods. Java does this to prevent you from calling methods that might not exist at runtime – for example, what if `c` happens to be a `HashSet` and you called `c.sort()`?

Java follows simple rules (think: "Java is dumb"). Even when it's clear to you that `c` here is definitely an `ArrayList`, you still have to declare it as such. That is,

```
Collection c = new ArrayList();
c.sort();
```

will still fail at compile time. This is not necessarily a bad thing! When I declare `c` to be a `Collection` here, it kind of means I'm saying "I just want a `Collection`, it'll be an `ArrayList` here but I don't want to do any `ArrayList`-specific things."

```
// Example 3: works, but has different results
c.add(1);
c.add(1);
c.size();  // Will this equal 1 or 2?
```

Note that `Collection` has no method implementation of its own. Java knows to look at the methods for `HashSet` or `ArrayList`, depending on what the dynamic type of `c` is.

4. Static and dynamic types questions

4a. Spot the compile time errors. (There are four!)

File: CompileTimeErrorTest.java

```
public class CompileTimeErrorTest {
    public string howOld(age) {
//          ^~~~~ needs to be capitalized
//                      ^~~~~ missing "int"
        if age <= 18 {
//          ^~~~~~~~ missing parentheses
            return "Not very old";
        } else if (age > 21) {
            return "Really old";
        }
//      ^ missing unconditional or "else" return statement
    }
}
```

4b. Where is the runtime error?

File: RuntimeErrorTest.java

```
public class RuntimeErrorTest {
    private Person p;

    public RuntimeErrorTest() {
        String personName = p.getName();
        int nameLength = personName.length();
        System.out.println(nameLength);
    }

    public static void main(String[] args) {
        RuntimeErrorTest t = new RuntimeErrorTest();
    }
}

class Person {
    public String getName() {}
}
```

5. Vroom Vroom!

To get the car rolling!

File: Vehicle.java

```
import java.util.ArrayList;

public abstract class Vehicle {
    int seats;
    int wheels;
    int fuel;
    int mpg;
    int trunkSize;
    ArrayList<Object> trunk;

    public Vehicle(int seats, int wheels, int fuel, int mpg) {
        this.seats = seats;
        this.wheels = wheels;
        this.fuel = fuel;
        this.mpg = mpg;
        this.trunk = new ArrayList<Object>();
        this.trunkSize = 0;
    }

    public void putInTrunk(Object item) {
        System.out.println("There is no room in the Trunk");
    }

    float range() {
        return fuel * mpg;
    }
}

class Car extends Vehicle {
    public Car(int fuel, int mpg) {
        super(4, 4, fuel, mpg);
        this.trunkSize = 2;
    }

    public void putInTrunk(Object item) {
        if (this.trunk.size() < this.trunkSize) {
            trunk.add(item);
        } else {
            super.putInTrunk(item);
        }
    }
}


class Motorcycle extends Vehicle {
    public Motorcycle(int fuel, int mpg) {
        super(1, 2, fuel, mpg);
    }
}
```

```
/* Fill this class in assuming that the trunkSize of a Truck is 5*/
public class Truck extends Car {
     public Truck() {

     }
}
```

What will happen after each of these snippets of code are compiled/run?

```
        //Q1
    Vehicle v1 = new Vehicle(3,4,20,10);
    System.out.println("Range of v1: " + v1.range());

    //Q2
    Vehicle v2 = new Car(20,20);
    System.out.println("Range of v2: " + v2.range());

    //Q3
    Vehicle v3 = new Motorcycle(10,40);
    System.out.println("Range of v3: " + v3.range());

    //Q4
    System.out.println("Number of seats of v2 " + v2.seats);
    System.out.println("Number of seats of v3 " + v3.seats);

    //Q5
    System.out.println("Number of wheels of v2" + v2.wheels);
    System.out.println("Number of wheels of v3" + v3.wheels);

    //Q6
    v2.putInTrunk("Backpack");
    v2.putInTrunk("Laptop");
    v2.putInTrunk("Shoes");

    //Q7
    v3.putInTrunk("Backpack");
    v3.putInTrunk("Laptop");
    v3.putInTrunk("Shoes");
```

Q1. Won't compile
Q2. 400
Q3. 400
Q4. 4, 1
Q5. 4, 2
Q6. It will print out "There is no room in the Trunk" once because of the third item.
Q7. It will print out "There is no room in the Trunk" three times because a Motorcycle has no trunk.