Final Review

This or That

1. Sorted data is required for which searching algorithm?
   (a) Sequential Search
   (b) Binary Search ✓
2. What is a composition?
   (a) Building a new class using instance variables of previously defined class ✓
   (b) Building a new class by extending from previous defined class
3. Ad hoc polymorphism is done through
   (a) overriding
   (b) overloading ✓
4. Java interface can be thought of as an
   (a) abstract sub-class with no instance data
   (b) abstract super-class with no instance data ✓
5. If an exception will not be handled in a method, the programmer have to use
   (a) try/catch block
   (b) throws statement ✓
6. A method that calls itself is called
   (a) recursive method ✓
   (b) static method

1 True/False

Indicate whether each of the following statements is true or false. For false statements explain WHY they are false.

1. Java instance variables are declared to be private and instance methods are declared to be public.

   Solution: False these are only guidelines, not requirements
2. Java subclass reference can be used to access superclass object

   Solution: False It is the other way around (superclass references can access subclass objects)
3. Instance methods initially defined in a subclass cannot be accessed through a superclass reference

   Solution: True

4. A finally block of a Java exception handler is only executed if an exception occurs in the try block

   Solution: False the finally block is always executed

5. If I am sorting an array of objects that implement the Comparable interface, I do not care about anything in the objects except the `compareTo()` method

   Solution: True

2 Short Answers

1. In lecture, we discussed both **overloading** and **overriding** Java methods. Explain what each of these is and how they differ?

   Solution:

   - **Overloading**: methods within the same class (or in a common hierarchy) share the same name but have different method signature (parameters).
   - **Overriding**: method defined in a superclass is redefined in a subclass with identical method signature

2. In Java, we cannot create any objects of an abstract class. Therefore, why do we even have abstract classes? Explain in detail

   Solution: An abstract classes (inherited by subclasses) is defined simply to give cohesion to its subclasses.

3 Tracing Code

1. Give all output produced by the execution of the Java program below:

```
class AThing
{
        String name;
        public AThing(String aName)
        {
                name = aName;
        }
        public String toString()
        {
                return "A Thing: " + name;
        }
}

class TheThing extends AThing
{
        int height;
        public TheThing(String aName, int aHeight)
        {
                super(aName);
                height = aHeight;
        }
        public String toString()
        {
                return "The Thing: " + height + "-feet " + name;
        }
}

public class Trace1
{
        public static void main(String[] args)
        {
                AThing a = new AThing("Animal");
                System.out.println(a);
                TheThing b = new TheThing("Monster", 7);
                System.out.println(b);
                a = b;
                System.out.println(a);
        }
}
```

Solution:

```
A Thing: Animal
The Thing: 7-feet Monster
The Thing: 7-feet Monster
```

2. Give all output produced by the execution of the Java program below. **HINT**: Note that the program is recursive

```
public class Trace2
{
    public static int dualMove(int left, int right)
    {
        System.out.println("Left: " + left + " Right: " + right);
        if(left <= right)
        {
            System.out.println("Recursing...");
            int answer = dualMove(left + 1, right - 1) + left + right;
            System.out.println("Returning " + answer);
            return answer;
        }
        else
        {
            int answer = left + right;
            System.out.println("Base case returns " + answer);
            return answer;
        }
    }

    public static void main(String[] args)
    {
        int low = 2, high = 7;
        int answer = dualMove(low, high);
        System.out.println("The answer is " + answer);
    }
}
```

Solution:

```
Left: 2 Right: 7
Recursing...
Left: 3 Right: 6
Recursing...
Left: 4 Right: 5
Recursing...
Left: 5 Right: 4
Base case returns 9
Returning 18
Returning 27
Returning 36
The answer is 36
```

4 Coding

1. Write a single complete Java program that does the following:

   (a) Prompts the user to enter an integer and reads it in as a `String`

   (b) Converts the `String` into an actual `int`

   (c) If any errors occur in the conversion, tells the user about it, then prompts the user to enter another `String`, and repeats this process until a valid `int` has been entered

   (d) Output the result onto the display

   See the example output below. **Note** that you must use an exception handler to implement this program.

```
Please enter an integer
bogus
Your data was invalid
Please enter an integer
weasel
Your data was invalid
Please enter an integer
34.45
Your data was invalid
Please enter an integer
234
You entered: 345
```

   Solution: Answers will vary. One possibility is shown below.

```java
import java.util.Scanner;

public class Prac2Code1
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        String aData = null;
        int aValue = 0;
        boolean valid = false;

        while(!valid)
        {
            System.out.println("Please enter an integer");
            aData = scan.nextLine();
            try
            {
                aValue = Integer.parseInt(aData);
                valid = true;
            }
            catch (NumberFormatException e)
            {
                System.out.println("Your data was invalid");
            }
        }
        System.out.println("You entered: " + aValue);
    }
}
```

2. Consider the main program and its output below:

```
public class Prac2Code2
{
    public static void main(String [] args)
    {
        Grocery[] items = new Grocery[4];
        items[0] = new BulkItem("Sugar", 5.0, 0.25);
        items[1] = new SingleItem("Eggs", 1.15);
        items[2] = new BulkItem("Beef", 3.0, 2.50);
        items[3] = new SingleItem("Soup", 1.75);

        for(int i = 0; i < items.length; i++)
        {
            System.out.println("Product: " + items[i] + " Cost: " + items[i].cost());
        }
    }
}
```

Output:

```
Product: Sugar Quantity: 5.0 Cost: 1.25
Product: Eggs Cost: 1.15
Product: Beef Quantity: 3.0 Cost 7.5
Product: Soup Cost: 1.75
```

Write classes `Grocery`, `BulkItem`, and `SingleItem` such that the program will execute as indicated. You must abide by the following restrictions:

(a) Class `Grocery` must be **abstract**

(b) The product name must be declared in class `Grocery` and it must be a private instance variable.

Solution:

Grocery.java

```
public abstract class Grocery
{
    private String name;

    public Grocery(String aName)
    {
        name = aName;
    }
    public abstract double cost();
    public String toString()
    {
        return name;
    }
}
```

BulkItem.java

```
public BulkItem extends Grocery
{
    private double quantity;
    private double costPer;
    public BulkItem(String aName, double q, double c)
    {
        super(aName);
        quantity = q;
        costPer = c;
    }
    public String toString()
    {
        String s = super.toString() + " Quantity: " + quantity;
    }
    public double cost()
    {
        return quantity * costPer;
    }
}
```

SingleItem.java

```
class SingleItem extends Grocery
{
    private double price;
    public SingleItem(String s, double c)
    {
        super(s);
        price = c;
    }
    public double cost()
    {
        return price;
    }
}
```