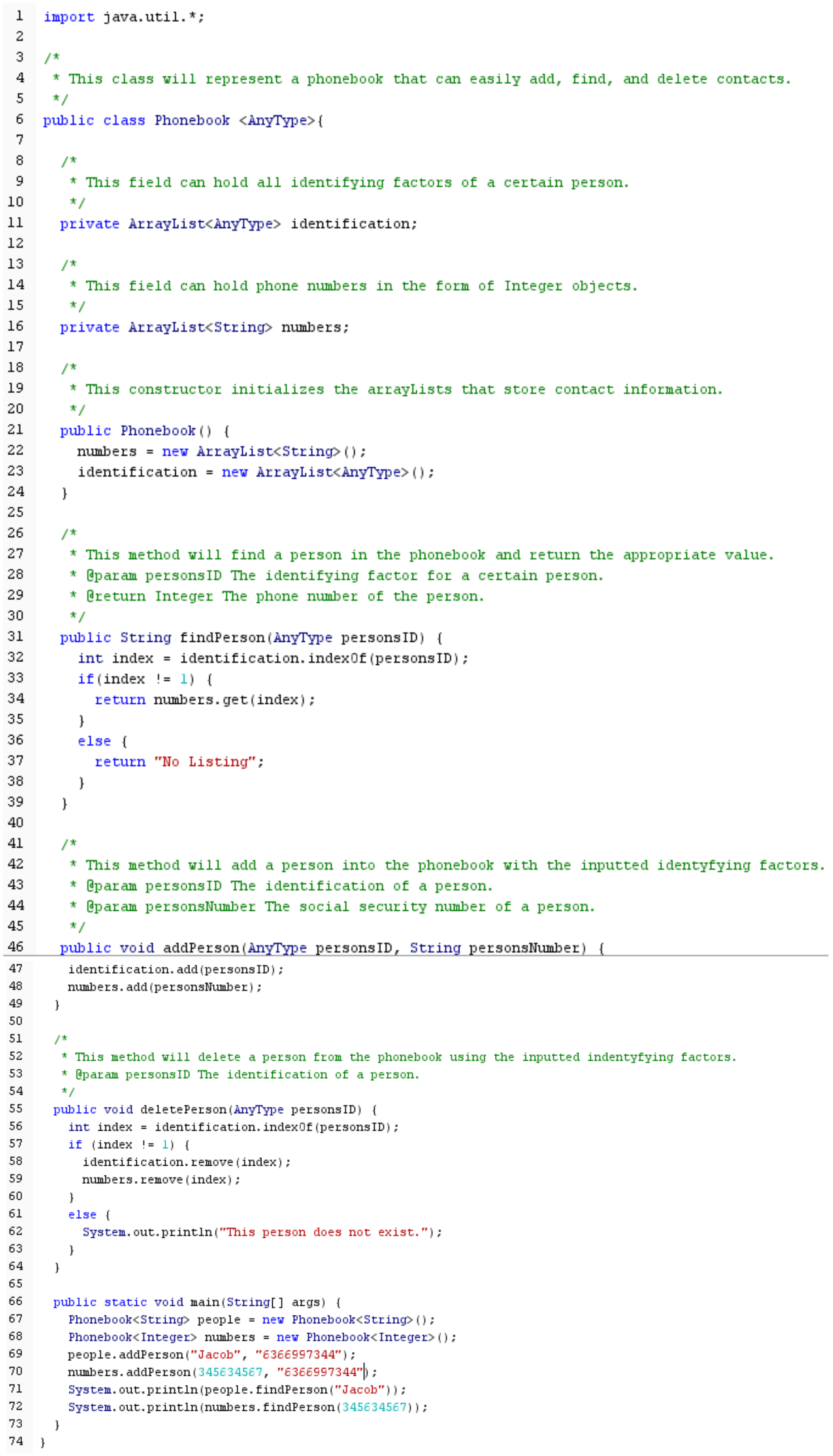
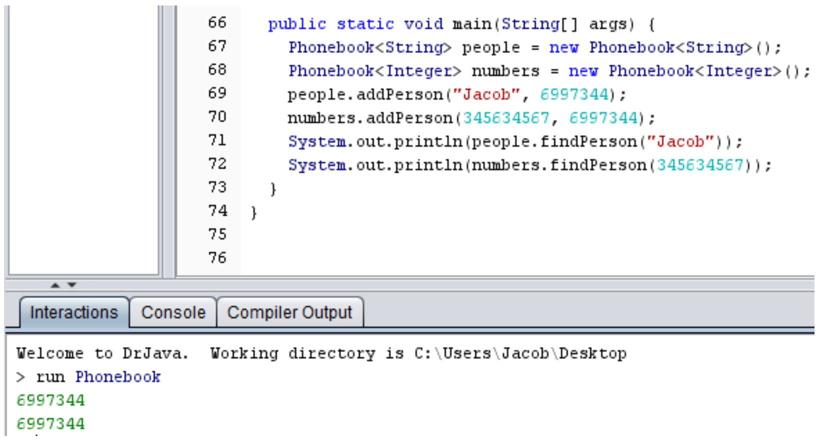
1)



```java
import java.util.*;

/*
 * This class will represent a phonebook that can easily add, find, and delete contacts.
 */
public class Phonebook <AnyType>{

  /*
   * This field can hold all identifying factors of a certain person.
   */
  private ArrayList<AnyType> identification;

  /*
   * This field can hold phone numbers in the form of Integer objects.
   */
  private ArrayList<String> numbers;

  /*
   * This constructor initializes the arrayLists that store contact information.
   */
  public Phonebook() {
    numbers = new ArrayList<String>();
    identification = new ArrayList<AnyType>();
  }

  /*
   * This method will find a person in the phonebook and return the appropriate value.
   * @param personsID The identifying factor for a certain person.
   * @return Integer The phone number of the person.
   */
  public String findPerson(AnyType personsID) {
    int index = identification.indexOf(personsID);
    if(index != 1) {
      return numbers.get(index);
    }
    else {
      return "No Listing";
    }
  }

  /*
   * This method will add a person into the phonebook with the inputted identyfying factors.
   * @param personsID The identification of a person.
   * @param personsNumber The social security number of a person.
   */
  public void addPerson(AnyType personsID, String personsNumber) {
    identification.add(personsID);
    numbers.add(personsNumber);
  }

  /*
   * This method will delete a person from the phonebook using the inputted indentyfying factors.
   * @param personsID The identification of a person.
   */
  public void deletePerson(AnyType personsID) {
    int index = identification.indexOf(personsID);
    if (index != 1) {
      identification.remove(index);
      numbers.remove(index);
    }
    else {
      System.out.println("This person does not exist.");
    }
  }

  public static void main(String[] args) {
    Phonebook<String> people = new Phonebook<String>();
    Phonebook<Integer> numbers = new Phonebook<Integer>();
    people.addPerson("Jacob", "6366997344");
    numbers.addPerson(345634567, "6366997344");
    System.out.println(people.findPerson("Jacob"));
    System.out.println(numbers.findPerson(345634567));
  }
}
```

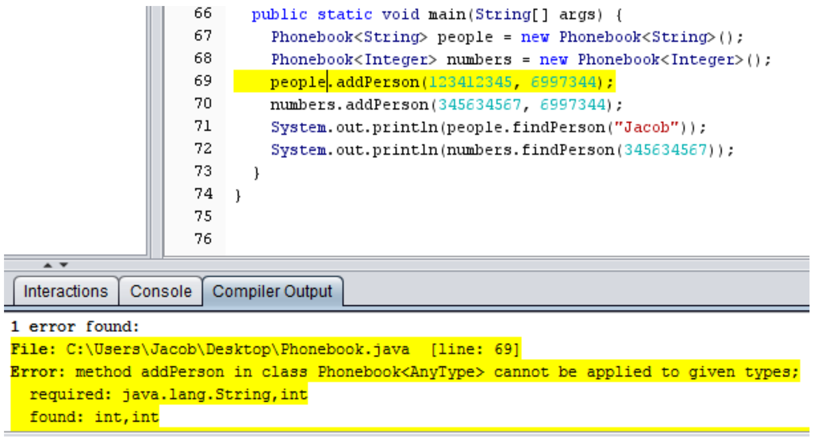


```
  public static void main(String[] args) {
    Phonebook<String> people = new Phonebook<String>();
    Phonebook<Integer> numbers = new Phonebook<Integer>();
    people.addPerson("Jacob", 6997344);
    numbers.addPerson(345634567, 6997344);
    System.out.println(people.findPerson("Jacob"));
    System.out.println(numbers.findPerson(345634567));
  }
}
```

Interactions | Console | Compiler Output

```
Welcome to DrJava.  Working directory is C:\Users\Jacob\Desktop
> run Phonebook
6997344
6997344
```

The output in the interaction pane is the correct inputted phone numbers.



```
  public static void main(String[] args) {
    Phonebook<String> people = new Phonebook<String>();
    Phonebook<Integer> numbers = new Phonebook<Integer>();
    people.addPerson(123412345, 6997344);
    numbers.addPerson(345634567, 6997344);
    System.out.println(people.findPerson("Jacob"));
    System.out.println(numbers.findPerson(345634567));
  }
}
```

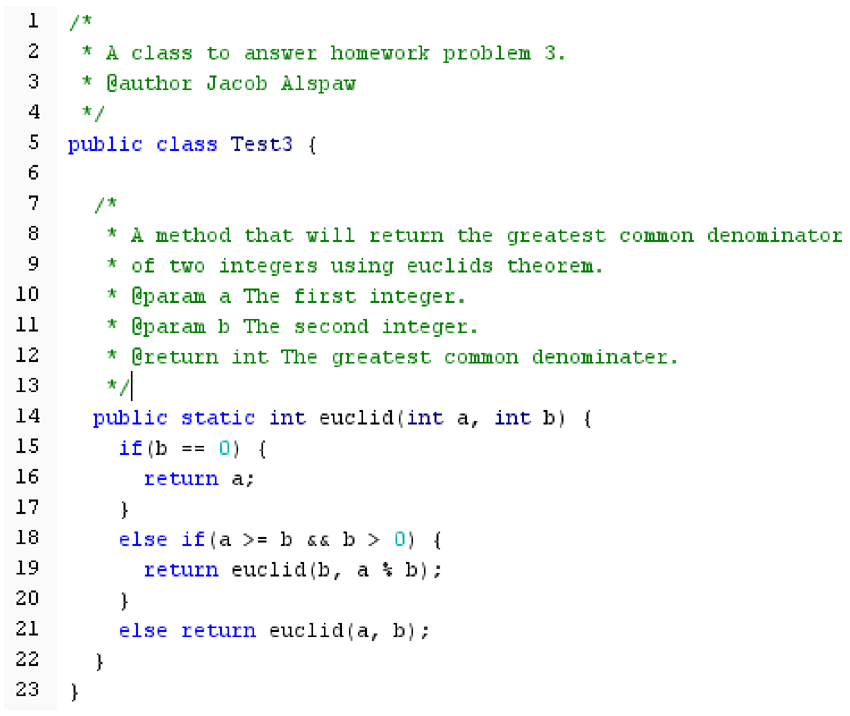
Interactions | Console | Compiler Output

```
1 error found:
File: C:\Users\Jacob\Desktop\Phonebook.java  [line: 69]
Error: method addPerson in class Phonebook<AnyType> cannot be applied to given types;
  required: java.lang.String,int
  found: int,int
```

The input for the phone book was of incorrect type and produced an error with this attempt.

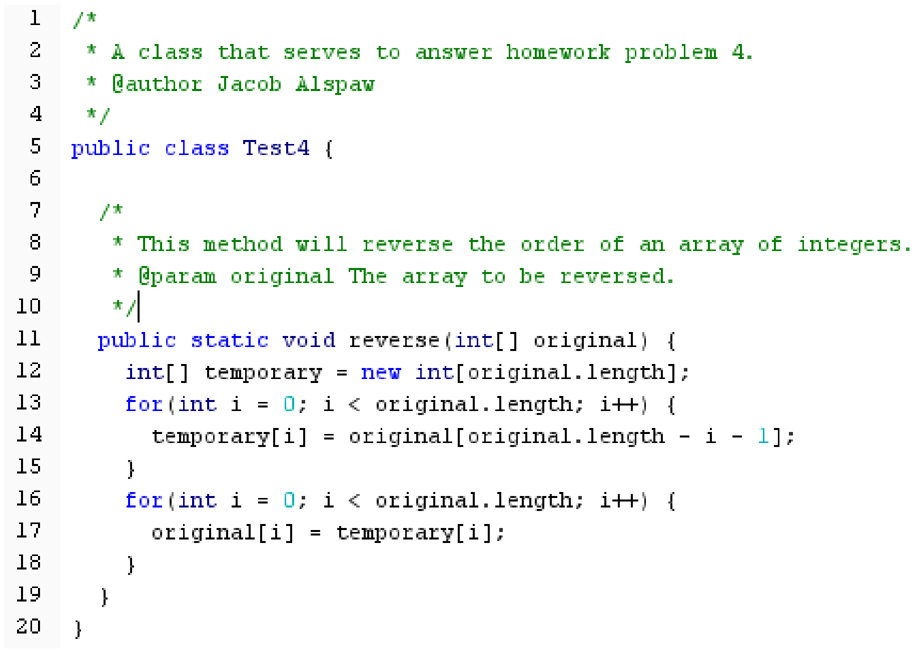
2) Question two will not work as a substitute for the method we discussed in class. The method will never meet the base case requirements of a string length of zero. The minimum length will always be one when a char is found. The recursion will run infinitely, no matter where the char is in the string and, in doing so, will produce stack overflow errors.

3)



```java
/*
 * A class to answer homework problem 3.
 * @author Jacob Alspaw
 */
public class Test3 {

  /*
   * A method that will return the greatest common denominator
   * of two integers using euclids theorem.
   * @param a The first integer.
   * @param b The second integer.
   * @return int The greatest common denominater.
   */
  public static int euclid(int a, int b) {
    if(b == 0) {
      return a;
    }
    else if(a >= b && b > 0) {
      return euclid(b, a % b);
    }
    else return euclid(a, b);
  }
}
```

4)



```java
/*
 * A class that serves to answer homework problem 4.
 * @author Jacob Alspaw
 */
public class Test4 {

  /*
   * This method will reverse the order of an array of integers.
   * @param original The array to be reversed.
   */
  public static void reverse(int[] original) {
    int[] temporary = new int[original.length];
    for(int i = 0; i < original.length; i++) {
      temporary[i] = original[original.length - i - 1];
    }
    for(int i = 0; i < original.length; i++) {
      original[i] = temporary[i];
    }
  }
}
```

5) f(N) = O(g(N)) is equivalent to f(N) $\leq$ (constant)*g(N) by the definition of Big-O.

g(N) = O(h(N)) is equivalent to g(N) $\leq$ (constant)*h(N) by the definition of Big-O.

Therefore, due to the transitive property, f(N) $\leq$ (constant)*h(N). And by definition of Big-O, is equivalent to f(N) = O(h(N)).

6) a. O(sqrt(n)) because sqrt(n) grows much faster.

b. O(log n) because of change of base.

c. O(n^3) because n^3 grows much faster.

7) a. Keeps track and returns the number of zeroes in an array. The Big-O bound of the worst-case running time is O(n). One for-loop will produce only a “n” number of iterations. The problem size is one iteration through the array. It is a linear size, so the length to the array is the length of problem size.

b. Returns the position of the first instance of a value of zero in an array. The Big-O bound of the worst-case running time is O(n). One for loop will produce only a “n” number of iterations. The problem size is again linear. However, do to variability of the array, the problem size will only be equal to or less than the size of the array.

c. Sorts an array in an order of smallest to largest values. The Big-O bound of the worst-case running time is O(n^2). Two for loops running through a designated size will be represented as “n^2” in a worst case scenario. The problem size is quadratic. The problem size for this case is the number of iterations for the for-loops multiplied by each other.