CS159 – Advanced Programming **James Madison University**
Sample Final Examination **Fall 2024**

Name:_____

Instructions. Answer all of the following questions. This is a "closed book" and you must work entirely on your own. You may not use a computing or communications device of any kind. You may make use of the attached Java Reference Card (in UML).

Your answers need not conform to the course style guide. All questions that involve code or are about code use the Java programming language.

This work complies with the JMU Honor Code.

Signature:_____

1. Given the following `Person` class:

```
public class Person {
    public int       age;
    public Person    bestFriend;
    public String    name;

    public Person(String name, int age, Person bestFriend) {
       this.name       = name;
       this.age        = age;
       this.bestFriend = bestFriend;
    }

    public String toString() {
       String     s;

       s = name + " (" + age + ") ";
       if (bestFriend == null) s += "has no friends";
       else                    s += "is friends with " + bestFriend.name;
       return s;
    }
}
```

what will be printed by the following `Driver`? **Note: Be careful!**

```
import java.util.*;

public class Driver {

    public static void main(String[] args) {
       Person       alice, bob, carol, dan, temp;
       ArrayList    people;

       alice = new Person("Alice", 30, null);
       bob   = new Person("Bob",   32, alice);

       System.out.println(alice.toString()); // 1.1
       System.out.println(bob.toString());   // 1.2

       carol = new Person("Carol", 40, null);
       dan   = new Person("Dan",   50, carol);
       carol.bestFriend = dan;

       System.out.println(carol.toString());  // 1.3
       System.out.println(dan.toString());    // 1.4

       people = new ArrayList();
       people.add(alice);
       people.add(bob);
       people.add(carol);
       people.add(dan);

       alice.age      = 65;
       bob.bestFriend = carol;
       carol.name     = "Carol Ann";
       dan.age        = 88;

       for (int i=0; i<people.size(); i++) {
          temp = (Person)people.get(i);
          System.out.println(temp.toString()); // 1.5, 1.6, 1.7, 1.8
       }
    }
}
```

1.1_____

1.2_____

1.3_____

1.4_____

1.5_____

1.6_____

1.7_____

1.8_____

2. Given the following `Filler` class:

```
public class Filler {

    private int[][] table = {
            {0, 0, 0},
            {0, 0, 0},
            {0, 0, 0}
    };

    public Filler() {
    }

    public int fill(int row, int col, int value) {

        table[row][col] = value;

        if ((row-1) >= 0 && table[row-1][col] == 0)
            value = fill(row-1, col, value+1);


        if ((col+1) <  3 && table[row][col+1] == 0)
            value = fill(row, col+1, value+1);


        if ((row+1) <  3 && table[row+1][col] == 0)
            value = fill(row+1, col, value+1);


        if ((col-1) >= 0 && table[row][col-1] == 0)
            value = fill(row, col-1, value+1);


        return value;
    }
}
```

what values will be in the following "table" after the following statements are executed?

```
        Filler f= new Filler();
        f.fill(2,1, 1);
```

Columns

Rows	0	1	2
0			
1			
2			

3. Given the following interfaces and classes:

```
public interface Sized {
    public int getSize();
}
```

```
public class Business implements Sized {
    public String name;
    public int squareFeet;

    public Business(String name , int squareFeet) {
        this.name = name;
        this.squareFeet = squareFeet;
    }

    public int getSize() {
        return squareFeet;
    }

    public String toString() {
        return name + " (" + squareFeet + " sq. feet )";
    }
}
```

```
public abstract class Residence implements Sized {
    private int squareFeet;

    public Residence(int squareFeet) {
        this.squareFeet = squareFeet;
    }

    public abstract int payment();

    public int getSize() {
        return squareFeet;
    }

    public String toString() {
        return " Square feet : " + getSize() + " Payment : " + payment();
    }
}
```

```
public class Apartment extends Residence {
    private int rent;

    public Apartment(int rent ,int squareFeet)     {
        super(squareFeet);
        this.rent = rent ;
    }

    public int payment()     {
        return rent;
    }
}
```

```
public class Condo extends Residence {
    private int mortgage;

    public Condo(int mortgage, int squareFeet) {
        super(squareFeet);
        this.mortgage = mortgage ;
    }

    public int payment() {
        return mortgage ;
    }

    public String toString() {
        return super.toString() + " plus condo fees!";
    }
}
```

and the following declarations and instantiations:

```
    Sized[] stuff ;
    Residence[] homes ;
    Condo[] development ;

    stuff = new Sized[3];
    homes = new Residence[3];
    development = new Condo[3];
```

indicate whether each of the following statements will: not compile (N), compile but generate an exception at runtime (X), or compile and run without generating an exception (R).

3.1____`stuff[0] = new Business("Martins", 12000);`

3.2____`stuff[1] = new Residence(2200);`

3.3____`stuff[2] = new Condo(900, 2200);`

3.4____`stuff[0] = homes[0];`

3.5____`homes[0] = stuff[0];`

3.6____`stuff[0] = development[0];`

3.7____`homes[0] = new Business("Kroger", 10000);`

3.8____`homes[1] = new Residence(2200);`

3.9____`homes[2] = new Apartment(600, 1200);`

3.10____`development[0] = new Condo(600, 1200);`

3.11____`stuff[0].name = "Food Lion";`

```
3.12____development[0].squareFeet = 200;

3.13____int p = homes[0].payment();

3.14____int q = development[0].payment();

3.15____int m = development[0].mortgage;

3.16____int s = stuff[0].getSize();

3.17____int t = homes[0].getSize();
```

4. Given the `Sized` interface and the `Business`, `Residence`, `Condo` and `Apartment` classes above, consider the following `Relocation` class.

```
public class Relocation {

  public Residence    from, to;

  public Relocation(Residence from, Residence to) {
    this.from = from;
    this.to   = to;
  }

  public boolean isBetter() {
    return to.getSize() > from.getSize();
  }
}
```

4.1 Will the following fragment compile? If not, why not? If so, what output will it generate?

```
Residence   nj, va;

nj = new Condo(2000, 1000);
va = new Apartment(1200, 1100);

Relocation  move;

move = new Relocation(nj, va);
System.out.println(move.isBetter());
```

4.2 Will the following fragment compile? If not, why not? If so, what output will it generate?

```
Apartment va;
Condo nj;

nj = new Condo(2000, 1000);
va = new Apartment(1200, 1100);

Relocation  move;

move = new Relocation(nj, va);
System.out.println(move.toString());
```

5. Given the interfaces and classes above, for each question below, show what will be printed when the associated fragment (and only the associated fragment) is inserted at the location denoted by the comment.

```
public class Driver {

    public static void info(Business b) {
        System.out.println("Business");
    }

    public static void info(Condo c) {
        System.out.println("Condo");
    }

    public static void info(Residence r) {
        System.out.println("Residence");
    }

    public static void info(Sized s) {
        System.out.println("Sized");
    }


    public static void main(String[] args) {

       // Each fragment goes here
    }
}
```

```
Sized s;
s = new Business("Martins", 12000);
info(s);
System.out.println(s.toString());
```

5.1 _____

_____

```
Business b;
b = new Business("Martins", 12000);
info(b);
System.out.println(b.toString());
```

5.2 _____

_____

```
Residence r;
r = new Condo(600, 1200);
info(r);
System.out.println(r.toString());
```

5.3 _____

_____

```
Condo c;
c = new Condo(900, 2200);
info(c);
System.out.println(c.toString());
```

5.4 _____

_____

```
Sized s;
s = new Condo(900, 2200);
info(s);
System.out.println(s.toString());
```

5.5 _____

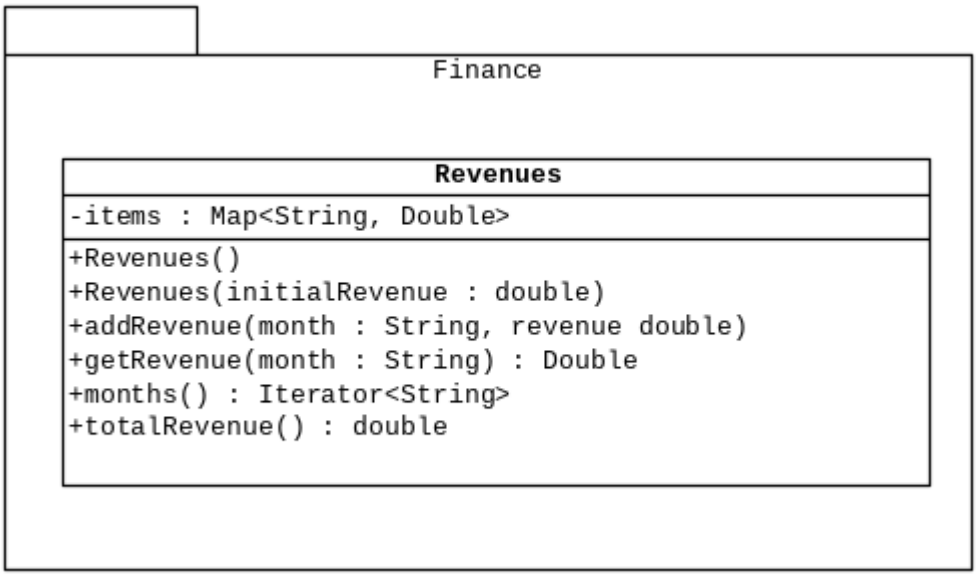
_____

```
Sized s;
s = new Apartment(600, 1200);
info(s);
System.out.println(s.toString());
```

5.6_____

_____

6. On the following page, complete the `Revenues` class summarized in the following UML class diagram.



In addition to the specifications in the UML diagram, your solution must satisfy the following specifications.

S1. All "collections" must be type-safe.

S2. The default constructor must initialize the attribute named `items`, but leave it empty.

S3. The constructor that is passed a `double` must add the months `"Jan"`, `"Feb"`, and `"Mar"` to the attribute named `items` with a `revenue` of `initialRevenue`.

S4. The `addRevenue()` method must increase the value for the given `month` by the given `revenue` if the given `month` is an existing key. Otherwise, it must insert the given key and value into the attribute named `items`.

S5. The `totalRevenue()` method must return the total revenue of all elements in the items attribute.

S6. The `months()` method must return an `Iterator` of all of the months for which there are revenues.

7. Given the import statements below, write a complete (i.e., nothing may be omitted) JUnit test of the `totalRevenues()` method in the `Revenues` class. You should assume that the `Revenues` class was implemented correctly.

A correct implementation of the `Revenues` class must pass your test of the `totalRevenues()` method when the expected value is 3010.50 and there are at least two months of revenues. In other words, your inputs to the tests must be such that the value returned by the `totalRevenues()` method in a correct implementation of the `Revenues` class is 3010.50 and there are at least two months of revenues.

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;
```

8. In the `Finance` class, suppose you wanted to keep track of multiple revenues/sales for each month, rather than keeping track of the total revenue for each month.

8.1 How would the declaration statement of the `items` attribute change?

8.2 Write the `addRevenue()` method for this situation.

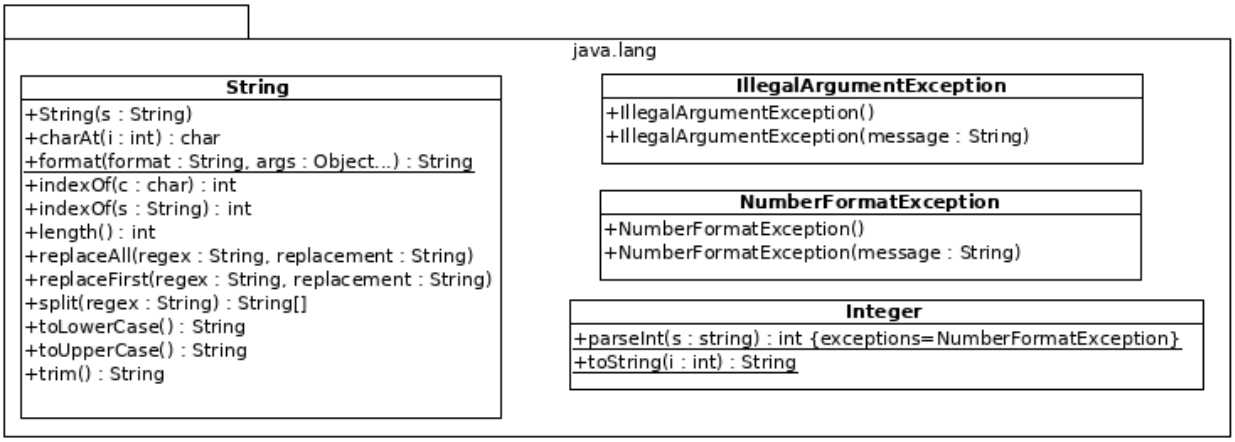
9. What would be printed bye the following `main()` method in the `Sports` class?

```
public class Sports {

    public static void main(String[] args) {

        int          game, points, week;
        int[][]      pointsScored;        // Note the declaration

        pointsScored    = new int[3][];

        pointsScored[0]    = new int[2];
        pointsScored[0][0] = 58;
        pointsScored[0][1] = 60;

        pointsScored[1]    = new int[4];
        pointsScored[1][0] = 72;
        pointsScored[1][1] = 48;
        pointsScored[1][2] = 61;
        pointsScored[1][3] = 44;


        pointsScored[2]    = new int[3];
        pointsScored[2][0] = 60;
        pointsScored[2][1] = 63;
        pointsScored[2][2] = 51;

        System.out.println(pointsScored[2].length);

        int total = 0;
        for (int i=0; i<pointsScored.length; i++) {
            total += pointsScored[i][0];
        }
        System.out.println(total);

        pointsScored[1] = new int[] {25};
        for (int i=0; i<pointsScored[1].length; i++) {
            System.out.println(pointsScored[1][i]);
        }
    }
}
```

10. Complete the following method that returns `true` if and only if the given word is a palindrome (i.e., if and only if the word reads the same backward and forward). For example, it must return `true` when passed the `String` `"racecar"`. `"civic"`, or `"noon"`, and must return `false` when passed the `String` `"computer"`. **Your solution must be recursive.** You may assume that all of the characters in the word will be lower case and that it will not contain punctuation characters or white space characters.

```
public static boolean isPalindrome(String word) {









}
```

Java Reference Card

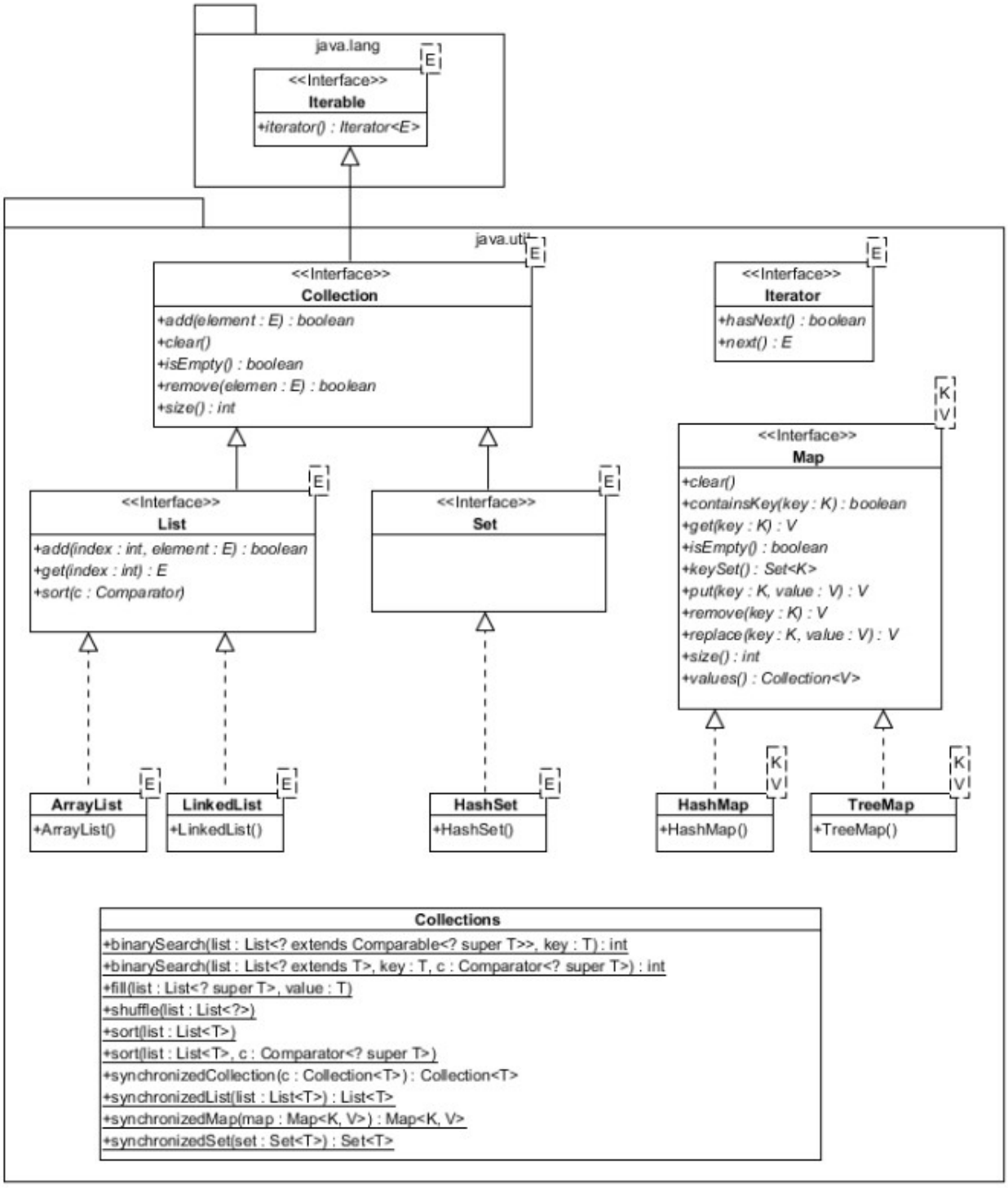


Methods Inherited by Every `class`

```
public boolean equals(Object other)
public int hashCode()
public String toString()
```

Methods Inherited by Every `enum`

```
public int compareTo(E other)
public boolean equals(E other)
public int ordinal()
public String toString()
public static E[] values()
public static E valueOf(String s) throws IllegalArgumentException
```


java.lang
E
<<Interface>>
Iterable
+iterator() : Iterator<E>
java.util
E
<<Interface>>
Collection
+add(element : E) : boolean
+clear()
+isEmpty() : boolean
+remove(elemen : E) : boolean
+size() : int
E
<<Interface>>
Iterator
+hasNext() : boolean
+next() : E
K
V
<<Interface>>
Map
+clear()
+containsKey(key : K) : boolean
+get(key : K) : V
+isEmpty() : boolean
+keySet() : Set<K>
+put(key : K, value : V) : V
+remove(key : K) : V
+replace(key : K, value : V) : V
+size() : int
+values() : Collection<V>
E
<<Interface>>
List
+add(index : int, element : E) : boolean
+get(index : int) : E
+sort(c : Comparator)
E
<<Interface>>
Set
E
ArrayList
+ArrayList()
E
LinkedList
+LinkedList()
E
HashSet
+HashSet()
K
V
HashMap
+HashMap()
K
V
TreeMap
+TreeMap()
Collections
+binarySearch(list : List<? extends Comparable<? super T>>, key : T) : int
+binarySearch(list : List<? extends T>, key : T, c : Comparator<? super T>) : int
+fill(list : List<? super T>, value : T)
+shuffle(list : List<?>)
+sort(list : List<T>)
+sort(list : List<T>, c : Comparator<? super T>)
+synchronizedCollection(c : Collection<T>) : Collection<T>
+synchronizedList(list : List<T>) : List<T>
+synchronizedMap(map : Map<K, V>) : Map<K, V>
+synchronizedSet(set : Set<T>) : Set<T>

JUnit Reference Card

This part of the reference card contains examples of JUnit tests that use static methods in the `org.junit.jupiter.api.Assertions` class.

An example of a JUnit test that uses the `assertEquals()` method:

```
@Test
public void getAtomicNumber() {
    Atom o;
    o = new Atom("O", 8, 16);
    assertEquals(8, o.getAtomicNumber(), "Oxygen");
}
```

An example of a JUnit test that doesn't use a lambda expression to test for an exception:

```
@Test
public void constructor_IllegalArguments() {
    try {
        new Atom("O", -8, -16);
        fail("Should have thrown an IllegalArgumentException");
    } catch (IllegalArgumentException iae) {
        // The exception was thrown as expected
    }
}
```

An example of a JUnit test that uses a lambda expression to test for an exception:

```
@Test
public void constructor_IllegalArguments() {
    assertThrows(IllegalArgumentException.class, () -> {
        new Atom("O", -8, -16);
    });
}
```

This part of the reference card contains other useful static methods in the `org.junit.jupiter.api.Assertions` class.

```
assertEquals(double expected, double actual, double tolerance)
assertFalse(boolean actual)
assertTrue(boolean actual)
assertNull(Object actual)
assertSame(Object expected, Object actual)
assertArrayEquals(Object[] expected, Object[] actual)
```