Exam #1 Doc# 1

VOCAB KEY

1 Encapsulation

2 Class

3 Polymorphism

4 Abstract Method

5 Instance

6 Variable

7 Definition

8 Getter

9 Setter

10 Object

11 Destructor

12 Operator

13 Declaration

14 Attribute

15 Constructor

VOCAB KEY

1 Reentrant

2 Validation Rules

3 Data Validation

4 Agile Process

5 Generic

6 Mutex

7 Generic Programming

8 Class Library

9 Stack

10 Class Hierarchy

11 Global

12 Thread

13 Code

14 Baseline

15 Shadowing

MULTIPLE CHOICE KEY

1 A	6 C	11 D	16 B
2 B	7 A	12 B	17 D
3 A	8 C	13 D	18 D
4 D	9 D	14 A	19 B
5 B	10 B	15 C	20 A

1. Generics {5 points}

```
public static <E> void print(E value) {
    System.out.print(value + " ");
}
```

2. Threads {8 points}

```
import java.util.ArrayList;

public class CritterLambda {
   private static Object mutex = new Object();
   private static ArrayList<String> sounds = new ArrayList<>();
   public static void chatter(String sound) {
       for(double f=0; f<Math.random()*6; ++f)
       synchronized(mutex) {sounds.add(sound);}
   }
   public static void main(String[] args) {
       ArrayList<Thread> threads = new ArrayList<>();
       String[] says = {"arf", "meow", "chirp", "quack", "moo",
                        "cluck", "hiss", "oink", "roar", "whinny"};

       for(String s: says)
           threads.add(new Thread(() -> chatter(s)));
       for(Thread t: threads)
           t.start();

       try {
           for(Thread t : threads) t.join();
       } catch (InterruptedException e) {
       }
       for(String s : sounds) System.out.println(s);
   }
}
```

Additional acceptable solution shown on next page.

```
import java.util.ArrayList;

public class CritterSync {
   private static ArrayList<String> sounds = new ArrayList<>();
   private synchronized static void addSound(String sound) {sounds.add(sound);}
   public static void chatter(String sound) {
       for(double f=0; f<Math.random()*6; ++f)
           addSound(sound);
   }
   public static void main(String[] args) {
       ArrayList<Thread> threads = new ArrayList<>();
       String[] says = {"arf", "meow", "chirp", "quack", "moo",
                        "cluck", "hiss", "oink", "roar", "whinny"};

       for(String s: says)
           threads.add(new Thread(() -> chatter(s)));
       for(Thread t: threads)
           t.start();

       try {
           for(Thread t : threads) t.join();
       } catch (InterruptedException e) {
       }
       for(String s : sounds) System.out.println(s);
   }
}
```

Additional acceptable solution shown on next page.

```java
import java.util.ArrayList;

public class CritterRunnable implements Runnable {
   private static Object mutex = new Object();
   private static ArrayList<String> sounds = new ArrayList<>();

   private String sound;
   public CritterRunnable(String sound) {this.sound = sound;}

   @Override
   public void run() {chatter(sound);}

   public static void chatter(String sound) {
       for(double f=0; f<Math.random()*6; ++f)
       synchronized(mutex) {sounds.add(sound);}
   }
   public static void main(String[] args) {
       ArrayList<Thread> threads = new ArrayList<>();
       String[] says = {"arf", "meow", "chirp", "quack", "moo",
                        "cluck", "hiss", "oink", "roar", "whinny"};

       for(String s: says)
           threads.add(new Thread(new CritterRunnable(s)));
       for(Thread t: threads)
           t.start();

       try {
           for(Thread t : threads) t.join();
       } catch (InterruptedException e) {
       }
       for(String s : sounds) System.out.println(s);
   }
}
```

3. Polymorphism. The virtual keyword with the area methods below is optional. The names of the variables may be whatever the student chooses.

-- 2, 2, and 3 points per code block, proportionately allocated based on validity

```
import java.util.ArrayList;

abstract class Shape {
   public abstract double area();
}

// Part a
class Circle extends Shape {
   //public static final double PI = 3.14159265;
   double radius;
   public Circle(double radius) {this.radius = radius;}
   @Override
   public double area() {return Math.PI * radius * radius;}
}

// Part b
class Rectangle extends Shape {
   double height;
   double width;
   public Rectangle(double height, double width) {this.height = height; this.width = width;}
   @Override
   public double area() {return height * width;}
}

// Part c
public class Shaper {
   public static void main(String[] args) {
       ArrayList<Shape> shapes = new ArrayList<>();
       shapes.add(new Circle(5));
       shapes.add(new Rectangle(3,4));
       for(Shape s : shapes) System.out.println(s.area());
   }
}
```

4. File I/O - 1½ points each proportionately allocated, 9 points total

a. {1½ points}

```
// QUESTION 4a - implement
public void save(BufferedWriter bw) throws IOException {
    bw.write("" + quantity + '\n');
    bw.write(name + '\n');
}
```

b. {1½ points}

```
// QUESTION 4b - implement
public Animal(BufferedReader br) throws IOException {
    quantity = Integer.parseInt(br.readLine());
    name = br.readLine();
}
```

c. {1½ points}

```
// QUESTION 4c - implement
public void save(BufferedWriter bw) throws IOException {
    bw.write("" + animals.size() + '\n');
    for(Animal a : animals) a.save(bw);
}
```

d. {1½ points}

```
// QUESTION 4d - implement
public Zoo(BufferedReader br) throws IOException {
    this();
    int size = Integer.parseInt(br.readLine());
    while(size-- > 0) animals.add(new Animal(br));
}
```

e. {1½ points}

```
// QUESTION 4e - implement to instance Zoo as zoo from file "zoo.txt"
//               Remember try-with-resources!
try (BufferedReader br = new BufferedReader(
                         new FileReader("zoo.txt"))) {
    zoo = new Zoo(br);
} catch (IOException e) {
    e.printStackTrace();
}
```

f. {1½ points}

```
// QUESTION 4f - implement to save Zoo instance zoo to file zoo.txt
//               Remember try-with-resources!
try (BufferedWriter bw = new BufferedWriter(
                         new FileWriter("zoo.txt"))) {
    zoo.save(bw);
} catch (IOException e) {
    e.printStackTrace();
}
```

5. Iterators {3 points}

```
// QUESTION 5 - Create the string representation USING ITERATORS
 @Override
 public String toString() {
     String result = "";
     Iterator it = animals.iterator();
     while(it.hasNext()) result += it.next().toString() + '\n';
     return result;
 }
```

6. Algorithms - 3 points for loading the data into the (we hope) TreeMap, 5 points for printing the states in sorted order, 8 points total

```
import java.util.TreeMap;
import java.util.Scanner;
import java.util.Arrays;

public class Counties {
   public static void main(String[] args) {
       Scanner in = new Scanner(System.in);
       TreeMap<String, Integer> counties = new TreeMap<>();
       while(in.hasNext())
           counties.put(in.nextLine(), Integer.parseInt(in.nextLine()));
       for(String state : counties.keySet())
           System.out.println(state + " " + counties.get(state));
   }
}
```

Bonus {+3 and +3 points}

a. This is a "guard". The state transition is blocked until the boolean inside the square brackets is true.

b. This is an "event". When an event occurs, an associated state transition may occur.