

Exam #2 Practice 4

VOCAB KEY

- 1 Concurrency
- 2 Thread
- 3 Heap
- 4 Branch
- 5 Generic Programming
- 6 Generic
- 7 Baseline
- 8 Abstraction
- 9 Exception
- 10 Process

MULTIPLE CHOICE KEY

1 A	6 B	11 D
2 C	7 D	12 D
3 A	8 C	13 A
4 B	9 C	14 A
5 D	10 A	15 D

FREE RESPONSE KEY

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.

```

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;
}
```

Bonus {+3 points} - Making field public would break encapsulation.