Full Name: _____

Student ID#: _____

CSE 1325 OBJECT-ORIENTED PROGRAMMING

PRACTICE #4 Exam #2 «---» 1 001 14 «---» Exam #2 PRACTICE #4

IMPORTANT: This practice exam is based on a 2½ hour final exam from a previous semester. It's length is NOT representative of the upcoming exam, but has been retained to give you extra practice Java coding techniques. Please allow yourself 2½ hours to complete this practice exam, or use it for untimed practice.

Instructions

1. Students are allowed pencils, erasers, and beverage only.
2. All books, bags, backpacks, phones, **smart watches**, and other electronics, etc. must be placed along the walls. **Silence all notifications.**
3. PRINT your name and student ID at the top of this page **and every coding sheet**, and verify that you have all pages.
4. **Read every question completely before you start to answer it.** If you have a question, please raise your hand. You may or may not get an answer, but it won't hurt to ask.
5. If you leave the room, you may not return.
6. You are required to SIGN and ABIDE BY the following Honor Pledge for each exam this semester.

NOTE: The number of questions in each section, and the topic of Free Response questions, may vary on the actual Final Exam.

Honor Pledge

On my honor, I pledge that I will not attempt to communicate with another student, view another student's work, or view any unauthorized notes or electronic devices during this exam. I understand that the professor and the CSE 1325 Course Curriculum Committee have zero tolerance for cheating of any kind, and that any violation of this pledge or the University honor code will result in an automatic grade of zero for the semester and referral to the Office of Student Conduct for scholastic dishonesty.

Student Signature: _____

WARNING: Questions are on the BACK of this page!

Vocabulary

Write the word or phrase from the Words list below to the left of the definition that it best matches. Each word or phrase is used at most once, but some will not be used. Each page has its own distinct word list. {10 at 2 point each}

Vocabulary

Word	Definition
1	Performing 2 or more algorithms (as it were) simultaneously
2	An independent path of execution within a process, running concurrently (as it appears) with other threads within a shared memory space
3	Memory shared by all threads of execution for dynamic allocation
4	A second distinct development path within the same organization and often within the same version control system
5	Writing algorithms in terms of types that are specified as parameters during instantiation or invocation
6	A Java construct representing a method or class in terms of generic types
7	A reference point in a version control system, usually indicating completion and approval of a product release and sometimes used to support a fork
8	Specifying a general interface while hiding implementation details
9	An object created to represent an error or other unusual occurrence and then propagated via special mechanisms until caught by special handling code
10	A self-contained execution environment including its own memory space

Word List

Abstraction	Algorithm	Baseline	Branch	Code
Concurrency	Encapsulation	Exception	Fork	Garbage Collector
Generic	Generic Programming	Global	Heap	Inheritance
Iterator	Multiple Inheritance	Mutex	OOP	Polymorphism
Process	Reentrant	Stack	Synchronized	Thread

Multiple Choice

Read the full question and every possible answer. Choose the one best answer for each question and write the corresponding letter in the blank next to the number. {20 at 1½ point each}

1. ____ **If an object has 3 synchronized methods named** `hop()`**,** `skip()`**, and** `jump()`**,**

   A. Only one thread may be executing *any* of these methods at a time (the methods are jointly protected)

   B. A separate mutex is still required to prevent thread interference

   C. Only a mutex may be synchronized, not a method

   D. Only one thread may be executing *each* method (the methods are separately protected)

2. ____ **To obtain the previous element from Java ListIterator it, write**

   A. `*it--`

   B. `*it`

   C. `it.previous()`

   D. A ListIterator cannot obtain the previous element

3. ____ **One difference between a Java Set and HashSet is**

   A. Set is the interface, HashSet (and TreeSet) implement Set

   B. A Set instance works with primitives, the HashSet with objects

   C. HashSet elements are accessed using two indices rather than one

   D. The HashSet elements are always sorted, Set elements are not

4. ____ **Subclass Orange extends superclass Fruit. Given** `Fruit f = new Orange();` **which demonstrates successful downcasting?**

   A. `Orange o = f;`

   B. `Orange o = (Orange) f;`

   C. `Orange o = new Orange(f);`

   D. Downcasting isn't possible with this code

5. ____ **If the generic type variable is not used in the body of a generic method, we may simply write**

   A. `public void shuffle(List<> list)`

   B. `public void shuffle(List<E> list)`

   C. `public <> void shuffle(List list)`

   D. `public void shuffle(List<?> list)`

6. ____ **Java allows generics to be defined as classes or**

    A. fields

    B. interfaces

    C. enums

    D. variables

7. ____ **Which is TRUE for a thread of execution in a Java program?**

    A. A Java thread runs as soon as its Thread object is instanced

    B. A thread is created by passing a method as the parameter to Thread's constructor

    C. "Busy loops" like `for(int i=0;i<100000;++i) ;` are a good way to pause a thread

    D. A thread shares memory with other threads within an operating system process

8. ____ **To open** `filename` **for reading, write**

    A. `FileReader f = new BufferedReader.createFileReader(filename);`

    B. `File f = fopen(filename, 'r');`

    C. `BufferedReader br = new BufferedReader(new FileReader(filename));`

    D. `File f = filename.open('r');`

9. ____ **To sort an ArrayList named list, write**

    A. `ArrayList.sort(list)`

    B. `list.sort()`

    C. `Collections.sort(list)`

    D. `// ArrayLists are always sorted`

10. ____ **Which method in superclass Super could be polymorphically called in its subclasses?**

    A. `public void bar()`

    B. `final void bar()`

    C. `void bar() poly`

    D. `private void bar()`

11. ____ **The reason to write a class or method as generic is**

A. To specify the declaration but supply the definition later

B. To use the preprocessor to adjust text of the program at compile time

C. To implement methods so they work differently based on their parameters

D. To allow it to be used with many different types

12. ____ **Which of the following will potentially pause a running thread?**

A. calling join on another thread to pause until it exits

B. calling Thread.sleep with the number of milliseconds to pause

C. calling a synchronized method

D. All of the above

13. ____ **The difference between an ArrayList and a HashMap is**

A. An ArrayList subscript is always an int, but a HashMap subscript may be almost any type

B. An ArrayList throws an exception if the subscript is out of range, but a HashMap does not

C. An ArrayList stores only primitive types, but a HashMap can story almost any type

D. The size of an ArrayList must be specified in advance, but a HashMap can change size dynamically

14. ____ **To ensure clean handling of I/O errors, use**

A. try-with-resources

B. ternary-try

C. try / catch / finally

D. nested try / catch as required

15. ____ **The difference betwen an Iterator and a ListIterator is**

A. ListIterator is a subclass of Iterator, and has more methods

B. ListIterator (unlike Iterator) is bi-directional, and can navigate forward and backward

C. ListIterator (unlike Iterator) can insert and overwrite items into the collection

D. All of these are differences between Iterator and ListIterator

Free Response

Provide clear, concise answers to each question. Each question may implement only a portion of a larger Java application. Each question, however, is *completely independent* of the other questions, and is intended to test your understanding of one aspect of Java programming. **Write only the code that is requested.** You will NOT write entire, large applications! **Additional paper is available on request.**

1. (generics) Consider the following code.

```
public class Output {
   public static void print(int value) {
       System.out.print(value + " ");
   }
   public static void main(String[] args) {
       for(int i=0; i<5; ++i) print(i);
       System.out.println();
   }
}
```

with output

```
0 1 2 3 4
```

Rewrite Output.print as a generic, such that the following main method

```
class Coordinate {
   private int x, y;
   public Coordinate(int x, int y) {this.x = x; this.y = y;}
   @Override public String toString() {return "(" + x + "," + y + ")";}
}
public class Output {
   // Add generic print method here
   public static void main(String[] args) {
       String[] s = {"one", "two", "three", "four", "five"};
       for(int i=0; i<5; ++i) {
           print(i);
           print(s[i]);
           print(new Coordinate(i*2, i*3));
       }
       System.out.println();
   }
}
```

produces the following output {5 points}:

```
0 one (0,0) 1 two (2,3) 2 three (4,6) 3 four (6,9) 4 five (8,12)
```

2. (threads) Consider the single-threaded application below. Modify in place, write only new or modified members, OR rewrite class Critter entirely (as you please) so that a separate thread runs each call to `Critter.chatter` concurrently.

a. {3 points} Ensure that adding a sound to the sounds ArrayList is thread-safe. You may select from several available options that we discussed in lecture.

b. {4 points} Run each call to chatter(s) in a separate thread

c. {3 points} Join all threads before printing the ArrayList {2 points}

```
import java.util.ArrayList;

public class Critter {


   private static ArrayList<String> sounds = new ArrayList<>();

   public static void chatter(String sound) {
       for(double f=0; f<Math.random()*6; ++f)
           // Question 2.a: protect ArrayList sounds

           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"};

       // Question 2.b: Run each call to chatter(s) in a separate thread
       for(String s: says)
           chatter(s);


       // Question 2.c: Join all threads before executing the following line


       for(String s : sounds) System.out.println(s);
   }
}
```

3. (polymorphism) Given the following base class:

```
import java.util.ArrayList;

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

a. Extend class Circle from Shape with a constructor that accepts a double radius, and that overrides method area() to calculate the area of the circle as π * radius * radius, where π is specified in Java as Math.PI. {2 points}

b. Extend class Rectangle from Shape with a constructor that accepts a double width and a double height, and that overrides method area() to calculate the area of the rectangle as width * height. {2 points}

c. Write a main method that stores at least one circle and one rectangle *in the same ArrayList*, and then in a for each loop prints the area of each to the console by polymorphically calling each object's area() method. Include all required import statements. {3 points}

Questions 4 and 5 are based on the following code, which implements a Zoo class containing an ArrayList of Animal instances. In question 4, you will write code to enable saving and loading an inventory of zoo animals from file zoo.txt. In question 5, you will use an Iterator to write Zoo.toString().

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;

public class Animal {
    private String name;  // The name of the animal, e.g. "giraffe"
    private int quantity; // The number of this animal in the zoo

    public Animal(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }

    // QUESTION 4b - implement
    public Animal(BufferedReader br) throws IOException {




    }

    // QUESTION 4a - implement
    public void save(BufferedWriter bw) throws IOException {




    }

    public String name() {return this.name;}
    public int quantity() {return this.quantity;}

    public void add(int quantity) {this.quantity += quantity;}
    @Override
    public String toString() {return quantity + " " + name;}
 }
```

```
  public class Zoo {
     private ArrayList<Animal> animals;
     public Zoo() {
         animals = new ArrayList<>();
     }

     // QUESTION 4d - implement
     public Zoo(BufferedReader br) throws IOException {


     }

     // QUESTION 4c - implement
     public void save(BufferedWriter bw) throws IOException {


     }
     public void addAnimals(int quantity, String name) {
         for (Animal a : animals) {
             if (a.name().equals(name)) {
                 a.add(quantity);
                 return;
             }
         }
         animals.add(new Animal(name, quantity));
     }

     // QUESTION 5 - Create Zoo's string representation USING ITERATORS
     @Override
     public String toString() {
         String result = "";


         return result;
     }
     public static void main(String[] args) {
         Zoo zoo = null;

         // QUESTION 4e - implement to instance Zoo as zoo from file "zoo.txt"
         //               Remember try-with-resources!


         zoo.addAnimals(2, "cheetahs");
         zoo.addAnimals(3, "sloths");
         System.out.println(zoo);

         // QUESTION 4f - implement to save Zoo instance zoo to file zoo.txt
         //               Remember try-with-resources!


     }
}
```

4. (File I/O) Based on the code above, implement the stream and file I/O code indicated. {1½ points each, 9 points total}

a. Implement Animal.save(BufferedWriter bw) such that class Animal's name and quantity is written to bw.

```
public void Animal.save(BufferedWriter bw) {
```

b. Implement Animal(BufferedReader br) such that the data written out in part a can now read from br to create an identical instance of Animal.

```
public Animal(BufferedReader br) {
```

c. Implement Zoo.save(BufferedWriter bw) such that class Zoo's private data is written to the output stream ost, delegating output to Animal::save(std::ostream& ost) as needed.

```
public void Zoo.save(BufferedWriter bw) {
```

d. Implement Zoo(BufferedReader br) such that the data written out in part a can now read from the input stream ist to create an identical instance of Zoo, delegating Animal construction to Animal::Animal(std::istream& ist) as required.

```
public Zoo(BufferedReader br) {
```

e. Open file "zoo.txt" for input, and construct a new instance of Zoo from it in a variable named zoo. Print the stack trace to STDERR on any IOException thrown.

f. Open file "zoo.txt" for output, and save the updated instance of Zoo in variable zoo to it. Print the stack trace to STDERR on any IOException thrown.

5. (Iterators) Implement Zoo.toString() *using an instance of class Iterator*. You will not receive any credit if you do not use Iterator effectively, since that is the purpose of this question. {3 points}

```
// QUESTION 5 - Create the string representation USING ITERATORS
 @Override
 public String toString() {
     String result = "";









     return result;
 }
```

Bonus: {+3 points} In one *concise* sentence, explain why we cannot simply make all of our class fields public and write single `save` and `load` methods just in the main class.