Full Name: ____

Student ID#: ___________________________

CSE 1325 OBJECT-ORIENTED PROGRAMMING

Final Exam Practice –=# 1 003 #=– Final Exam 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. {30 at 1 point each}

Vocabulary

Word	Definition
1	Bundling data and code into a restricted container
2	A template encapsulating data and code that manipulates it
3	The provision of a single interface to multiple derived classes, enabling the same method call to invoke different derived methods to generate different results
4	A method declared with no implementation
5	An encapsulated bundle of data and code
6	A block of memory associated with a symbolic name that contains a primitive data value or the address of an object instance
7	A declaration that also fully specifies the entity declared
8	A method that returns the value of a private variable
9	A method that changes the value of a private variable
10	An instance of a class containing a set of encapsulated data and associated methods
11	A special class member that cleans up when an object is deleted
12	A short string representing a mathematical, logical, or machine control action
13	A statement that introduces a name with an associated type into a scope
14	A class member variable (also called a field)
15	A special class member that creates and initializes an object from the class

Word List

Abstract Class	Abstract Method	Abstraction	Attribute	Class
Constructor	Declaration	Definition	Destructor	Encapsulation
Enumerated type	Garbage Collector	Getter	Inheritance	Instance
Interface	Method	Multiple Inheritance	Namespace	Object
Object-Oriented Programming (OOP)	Operator	Override	Package	Polymorphism
Primitive type	Setter	Subclass	Superclass	Variable

Vocabulary

Word	Definition
1	An algorithm that can be paused while executing, and then safely executed by a different thread
2	Algorithmically enforceable constraints on the correctness, meaningfulness, and security of input data
3	Ensuring that a program operates on clean, correct and useful data
4	Prioritizing individuals and interactions, frequent delivery of working software, customer collaboration, and flexible response to change
5	A Java construct representing a method or class in terms of generic types
6	An object that prevents two properly written threads from concurrently accessing a shared resource
7	Writing algorithms in terms of types that are specified as parameters during instantiation or invocation
8	A collection of classes designed to be used together efficiently
9	Scratch memory for a thread of execution
10	Defines the inheritance relationships between a set of classes
11	Memory for declarations outside of any class or function scope
12	An independent path of execution within a process, running concurrently (as it appears) with other threads within a shared memory space
13	Read-only memory for machine instructionsAlgorithms
14	A reference point in a version control system, using indicating completion and approval of a product release and sometimes used to support a fork
15	A variable declared in a narrower scope than that of a variable of the same name declared in a broader scope

Word List

Agile Process	Algorithm	Assertion	Baseline	Branch
Class Hierarchy	Class Library	Code	Concurrency	Data Validation
Exception	Fork	Generalization	Generic	Generic Programming
Global	Heap	Invariant	Iterator	Mutex
Process	Reentrant	Shadowing	Specialization	Stack
Synchronized	Thread	Validation Rules	Version Control	Waterfall Process

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. ____ **For dynamic polymorphism in Java to work,**

   A. a superclass variable must contain a subclass object with overridden methods

   B. a pointer variable must contain a subclass object

   C. a subclass variable must contain a superclass object with overridden methods

   D. the subclass method must be annotated with `@Override`

2. ____ **To advance iterator it to the next element in a collection, write**

   A. `next(it)`

   B. `it.next()`

   C. `Iterator.next(it)`

   D. `++it`

3. ____ **Given class Foo with method** `void f(String s)`**, to start a new thread running method f with parameter "Hello, Threads", write**

   A. `new Thread(() -> f("Hello, Threads"));`

   B. `new Thread(new Runnable(f("Hello, Threads"));`

   C. `new Thread(new Runnable -> f("Hello, Threads"));`

   D. `new Thread(f("Hello, Threads"));`

4. ____ **To use a class type with a SortedSet (HashSet) or as the key for a SortedMap (HashMap), override**

   A. equals and compareTo

   B. compareTo and toString

   C. clone

   D. equals and hashCode

5. ____ **Which of the following is NOT a subclass of interface Collection?**

   A. `Set`

   B. `Map`

   C. `Deque`

   D. `List`

6. ____ **To obtain the value for key "exam" in** `HashMap<String, Double> grades`**, write**

  A. `grades.find("exam")`

  B. `grades["exam"]`

  C. `grades.get("exam")`

  D. `grades.key("exam")`

7. ____ **For ArrayList v, to obtain an iterator pointing to element v[0], write**

  A. `v.iterator()`

  B. `v.begin()`

  C. `v.first()`

  D. `new ArrayList.iterator(v)`

8. ____ **To join a running thread named t1 to the current thread, write**

  A. `t1->join_thread();`

  B. `join(t1);`

  C. `t1.join();`

  D. `this->join_thread(t1);`

9. ____ **To remove all elements from ArrayList al, write**

  A. `for(var e : al) al.remove(e);`

  B. `al.clear();`

  C. `al.remove(al);`

  D. Any of these will work

10. ____ **To convert ArrayList<String> strings to an array of String, write**

  A. `String[] s; for(String v : strings) s.add(v);`

  B. `al.toArray()`

  C. `(String[]) al`

  D. `String[] s = new String[](v for v in strings)`

11. ____ **To compile a Java class Mandelbrot that uses threads, use the bash command**

   A. `javac -classpath Thread Mandelbrot.java`
   B. `javac +threads Mandelbrot.java`
   C. `javac -pthread Mandelbrot.java`
   D. `javac Mandelbrot.java`

12. ____ **Which is TRUE about Java lambdas?**

   A. A return statement is required
   B. If the value of the lambda expression is the return type, `return` may be omitted
   C. If no parameters are needed, the parentheses can be omitted
   D. Parameter types, if any, are required

13. ____ **To constrain a generic type to classes of type Integer or its superclasses, write**

   A. `void addNumbers(List<? superclass Integer> list)`
   B. `void addNumbers(List<? extends Integer> list)`
   C. `void addNumbers(List<? implements Integer> list)`
   D. `void addNumbers(List<? super Integer> list)`

14. ____ **A ListIterator is like an Iterator except**

   A. ListIterator is bidirectional and includes a previous() method, among others
   B. ListIterator works with Map (HashedMap)
   C. ListIterator is another name for Iterator - they are the same
   D. ListIterator works with a List (LinkedList)

15. ____ **Which is true about Java generics?**

   A. Generics may be declared with only a single generic type
   B. Generics must be defined in the Java Standard Library
   C. Generics are abstract
   D. Generics may only be instanced using primitive generic types

16. ____ **Which of these types is defined as a generic?**

   A. void
   B. ArrayList
   C. Thread
   D. int

17. ____ **To sort an ArrayList al, write**

   A. `al.sort();`
   B. `ArrayList<> alSorted = al.sort();`
   C. `ArrayList.sort(al);`
   D. `Collections.sort(al);`

18. ____ **Which method in class Super could be polymorphically called in its derived classes?**

   A. `void bar()`
   B. `@Override public void bar()`
   C. `private void bar()`
   D. All of the above

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

   A. enums
   B. methods
   C. variables
   D. interfaces

20. ____ `Thread.sleep(10)` **will**

   A. pause the current thread for 10 or more milliseconds
   B. join the current thread to the main thread in about 10 milliseconds
   C. pause the current thread for exactly 10 milliseconds
   D. create a new thread in under 10 milliseconds

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

b. Run each call to chatter(s) in a separate thread {3 points}

c. 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:

```java
import java.util.ArrayList;

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

a. Derive 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. Derive 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().

```java
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;}
 }
```

```java
  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;
 }
```

6. (Algorithms) Assume a text file containing an unsorted list of names of US states and territories and the number of counties (or equivalents) therein on separate lines, similar to this:

```
Ohio
88
Nebraska
93
New Mexico
33
Iowa
99
New Hampshire
10
```

And so on. This file will be provided to your program on STDIN, so that you may read it using Console or Scanner until end of file is reached. **Complete the code below.**

a. Read this file into a Map named states, with the state or territory name as the key. {3 points}

b. Print out the *sorted* name of each state or territory and the number of counties or equivalents that it has. {5 points}

The documentation for **Map** (which **HashMap** and **TreeMap** implement) is as follows. Default constructors may be used with any Map implementation.

void clear()

Removes all of the mappings from this map (optional operation).

V get(Object key)

Returns the value to which the specified key is mapped or null if this map contains no mapping for the key.

boolean isEmpty()

Returns true if this map contains no key-value mappings.

Set<K> keySet()

Returns a Set view of the keys contained in this map.

V put(K key, V value)

Associates the specified value with the specified key in this map (optional operation).

int size()

Returns the number of key-value mappings in this map.

Collection<V> values()

Returns a Collection view of the values contained in this map.

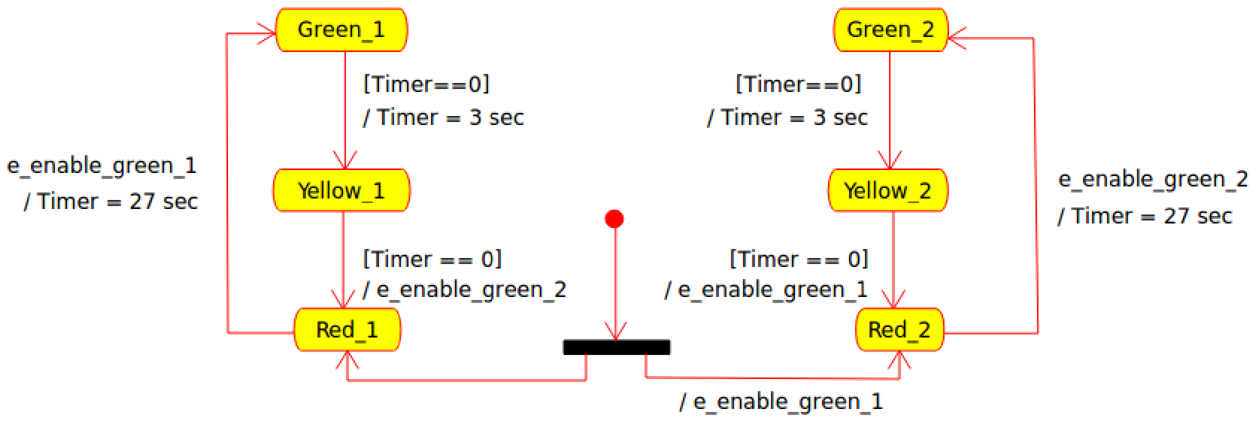
The **Collections.sort** method has the following documentation.

static <T extends Comparable<? super T>> void sort(List<T> list)

Sorts the specified list into ascending order according to the natural ordering of its elements.

Bonus

Consider the following state diagram.



a. What is the name for `[Timer == 0]` and what does it do? {+3 points}

b. What is the name for `e_enable_green_2` and what does it do? {+3 points}