Full Name: _____

Student ID#: _____

CSE 1325 OBJECT-ORIENTED PROGRAMMING

PRACTICE #2 Exam #2 «---» $BINARY 1 003 «---» Exam #2 PRACTICE #2

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.

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 Word 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. {15 at 2 points each}

Vocabulary

Word	Definition
1	A second distinct and independent development path undertaken (often by a different organization) to create a unique product
2	Scratch memory for a thread of execution (in Java, e.g., `int i=5;`)
3	A class that cannot be instantiated
4	A function that manipulates data in a class
5	Specifying a general interface while hiding implementation details (sometimes listed as a 4th fundamental concept of OOP, though I believe it's common to most paradigms)
6	Memory for static fields (and in C++, non-scoped variables)
7	A procedure for solving a specific problem, expressed as an ordered set of actions
8	The ability to control the access of multiple threads to any shared resource
9	A method declared with no implementation
10	A program that runs in managed memory systems to free unreferenced memory

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Baseline
Branch	Code	Concurrency	Data Validation	Encapsulation
Exception	Fork	Garbage Collector	Generic	Generic Programming
Global	Heap	Inheritance	Instance	Interface
Iterator	Method	Mutex	Polymorphism	Process
Reentrant	Reference Counter	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. {15 at 2 points each}

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

   A. `next(it)`

   B. `Iterator.next(it)`

   C. `++it`

   D. `it.next()`

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

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

   B. `t1->join_thread();`

   C. `join(t1);`

   D. `t1.join();`

3. ____ **Which Java code below MAY be polymorphic?**

   A. `students[3].grade();`

   B. `List<String> list = Arrays.asList(args); for(String s : list) sum += s.length(`

   C. `Student s = new Student("Alexander"); System.out.println(s);`

   D. `Student.newStudent();`

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

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

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

   C. a pointer variable must contain a subclass object

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

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

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

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

   C. `grades["exam"]`

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

6. ____ **To respect encapsulation while saving and loading program data,**

    A. delegate the reading and writing of each class's fields to that class

    B. add a `Struct save()` method to each class that returns a struct of its private data

    C. use `Object.pickle(object)` to automatically save each object's fields to the file

    D. create an I/O class in the package and set all fields to package-private

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

    A. `javac -classpath Thread Mandelbrot.java`

    B. `javac Mandelbrot.java`

    C. `javac +threads Mandelbrot.java`

    D. `javac -pthread Mandelbrot.java`

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

    A. Thread

    B. ArrayList

    C. int

    D. void

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

    A. `al.clear();`

    B. `al.remove(al);`

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

    D. Any of these will work

10. ____ **To read Strings from a text file opened as** `br` **until end of file is reached, write**

    A. `String[] lines = br.readLines(0, br.size());`

    B. `String line; for(int i=0; i<br.size(); ++i) line = br.readLine(i);`

    C. `String line; while((line = br.readLine()) != null)`

    D. Java can't detect end of file - use a special data value like "EOF!" instead

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

    A. `v.iterator()`
    B. `v.first()`
    C. `new ArrayList.iterator(v)`
    D. `v.begin()`

12. ____ **To declare a generic method, write**

    A. `public generic T getElement() <T>`
    B. `public generic <T> T getElement()`
    C. `public static <T> T getElement()`
    D. `public static T getElement() <T>`

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

    A. `String[] s; for(String v : strings) s.add(v);`
    B. `(String[]) al`
    C. `al.toArray()`
    D. `String[] s = new String[](v for v in strings)`

14. ____ **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(new Runnable -> f("Hello, Threads"));`
    B. `new Thread(() -> f("Hello, Threads"));`
    C. `new Thread(f("Hello, Threads"));`
    D. `new Thread(new Runnable(f("Hello, Threads"));`

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

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

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, map) In an aminal shelter, we may need to pair an Animal and the Client that adopted it. One approach might be a Pair class like this, stored in an ArrayList.

```
class Animal {
    /* more code */
    private String name;
    private Type type;
}
class Client {
    /* more code */
    private String name;
    private LocalDate birthday;
}
class Pair {
   public Pair(Animal animal, Client client) {
       this.animal = animal;
       this.client = client;
   }
   public Animal getAnimal() {return animal;}
   public Client getClient() {return client;}
   private Animal animal;
   private Client client;
}
class Shelter {
   private ArrayList<Pair> adoptions = new ArrayList<>();
   public void adopt(Animal animal, Client client) {
      adoptions.add(new Pair(animal, client));
   }
   // More code here
}
```

   a. {3 points} Many things come in pairs, so class `Pair` would be useful with *any* two generic types. In the space to the right of the code above, draw the UML class diagram representation of a **generic** class `Pair` using generic types K (for key) and V (for value). You may use `key` and `value` as parameter and field names in place of `animal` and `client`, and replace the getters with `getKey()` and `getValue()`.

   b. {6 points} Rewrite class `Pair` as a **generic** Java class storing an instance each of generic types K and V.

c. {3 points} Rewrite the following single line from class `Shelter` as a new single line that *properly* uses the new generic class `Pair` (that is, *without* a "uses unchecked or unsafe operation" compiler warning).

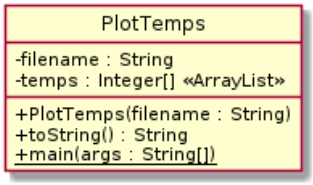
```
adoptions.add(new Pair(animal, client));
```

d. {5 points} Rather than generic class Pair, we might choose to use a HashMap to store Animal (the key) and Client (the value) objects. Rewrite the 3 lines shown above in class Shelter using a HashMap instead.

e. {3 points} To use class Animal as a key in a HashMap, we must override its `hashCode()` method. Write an overridden implementation of `hashCode()` for class Animal that includes both fields, while requiring the compiler to verify we have overridden a superclass method.

f. {6 points} Method `hashCode()` requires a compatible override of `equals(Object o)` as well. Write an implementation of method `equals(Object o)` for class Animal compatible with your implementation of `hashCode()` in the previous question. Two animals are equal if both their name and type are equal.

2. {file i/o, iterators} Consider the following program for printing a plot of recent temperatures here in Arlington. The text file whose filename is provided as the constructor parameter contains one integer (the temperature) per line.



Abbreviated example output:

```
Plot of Temps from 2023-10-09.txt

64 ================================================================>
63 ===============================================================>
63 ===============================================================>
61 ===============================================================>
```

Constructors

Constructor	Description
`IOException()`	Constructs an `IOException` with `null` as its error detail message.
`IOException(String message)`	Constructs an `IOException` with the specified detail message.
`IOException(String message, Throwable cause)`	Constructs an `IOException` with the specified detail message and cause.
`IOException(Throwable cause)`	Constructs an `IOException` with the specified cause and a detail message

a. {8 points} Finish the constructor. Open filename, ensuring it will be closed after reading. Read each line. If not empty, convert to an int and add to `temps`. If an IOException is thrown, throw it again, adding an appropriate message (see Constructors documentation).

```
public class PlotTemps {
    public PlotTemps(String filename) throws IOException {
        this.filename = filename;
```

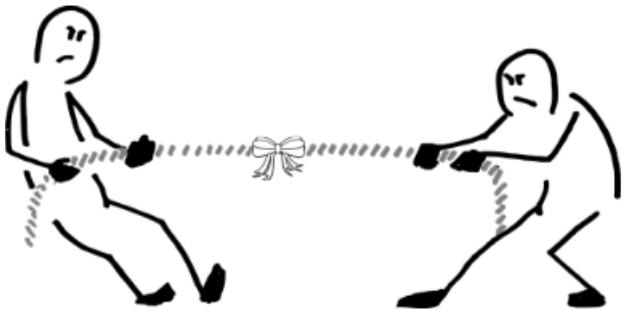
b. {4 points} Write the toString() method. Finish constructing the StringBuilder object, including the title + filename. **Obtain an iterator** from the `temps` ArrayList. **Using the iterator only,** obtain and append each temperature value to the StringBuilder object using format "%3d ", then append a number of "=" characters equal to the temperature value, and finally append ">\n". Once all temperatures have been added into the StringBuilder object, return its String representation.

```
private static String title = "Plot of Temperature from ";

@Override

public String toString() {
    StringBuilder sb =
```

3. (threads) In classic tug-of-war games, two teams pull on opposite ends of a rope with a ribbon tied to the center, trying to pull the ribbon to their side. (Image by dehaasbe per the Pixabay License.)



The code below simulates this game, where X marks the ribbon (see example output on the next page). We will rewrite method `tug()` to use 2 threads, one for each team, that repeatedly calls their pull method after a random delay until a team wins.

```
public class TugOfWar {
   private String rope = "------------X------------";
   private String winner = "";
   private void pullLeft() {
       rope = rope.substring(1) + rope.charAt(0);
       if(rope.charAt(0) == 'X' && winner.isEmpty())
           winner = "Left";
   }
   private void pullRight() {
       rope = rope.charAt(rope.length()-1) + rope.substring(0, rope.length()-1);
       if(rope.charAt(rope.length()-1) == 'X' && winner.isEmpty())
           winner = "Right";
   }
   public void tug() {
       while(winner.isEmpty()) {
           if(Math.random() < 0.5) pullLeft(); else pullRight();
           System.out.println(rope);
       }
       System.out.println(winner + " side wins!");
   }
   public static void main(String[] args) {
       (new TugOfWar()).tug();
   }
}
```

a. {4 points} Given that `pullLeft()` and `pullRight()` will be called asynchronously by different threads, what code would you change in these two methods to avoid thread interference? (You may state the change you would make, or rewrite part of a method to demonstrate your change.)

(Abbreviated) example output:

```
----X-------------------
-----X------------------
----X--------------------
---X---------------------
----X--------------------
---X---------------------
--X----------------------
---X---------------------
--X----------------------
-X-----------------------
X------------------------
Left side wins!
```

b. {8 points} Rewrite method `tug()` to use threads. You may use lambdas, anonymous classes, or class implementations of interface `Runnable` as you please. You may assume reading field `winner` is thread-safe in this context.

- Instance thread `left`, which loops until field `winner` is not empty, then exits. In the loop, call method `pullLeft()` and then sleep for 0 to 50 (randomly selected) milliseconds.
- Instance thread `right`, which is the same as thread `left` but calls method `pullRight()`. (You may abbreviate the body of thread `right` as just `...` to save time if you like - I'll know what you mean!)
- Start threads `left` and `right`.
- While field `winner` is empty, print field `rope` to the console every 1/10 of a second.
- Join all threads and print the winner.

Bonus

Bonus {+3 points} In no more than *two sentences*, explain hyperthreading.