Full Name: ____

Student ID#: ________________________

CSE 1325 OBJECT-ORIENTED PROGRAMMING

PRACTICE #3 Exam #2 «---» 1 1 001 «---» Exam #2 PRACTICE #3

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. {10 at 2 points each}

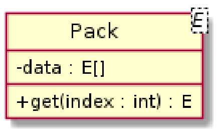
Vocabulary

Word	Definition
1	An object created to represent an error or other unusual occurrence and then propagated via special mechanisms until caught by special handling code
2	An independent path of execution within a process, running concurrently (as it appears) with other threads within a shared memory space
3	A style of programming focused on the use of classes and class hierarchies
4	A self-contained execution environment including its own memory space
5	A subclass inheriting class members from two or more superclasses
6	A second distinct development path within the same organization and often within the same version control system
7	A procedure for solving a specific problem, expressed in terms of an ordered set of actions to execute
8	An algorithm that can be paused while executing, and then safely executed by a different thread
9	Scratch memory for a thread of execution
10	A Java construct representing a method or class in terms of generic types

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

This class diagram is referenced in one or more Multiple Choice questions that follow.



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

   A. loops

   B. attributes

   C. enums

   D. methods

2. ____ **To obtain an Iterator to** `HashMap<String, String> map`**, write**

   A. `map.getIterator();`

   B. `map.mapIterator();`

   C. `map.listIterator();`

   D. Map implementations have no Iterator

3. ____ **Java class Pack in the UML diagram under Vocabulary is definitely**

   A. Extensible

   B. Generic

   C. Static

   D. Public

4. ____ **To define a generic class named Foo in Java, write**

   A. `generic class Foo <T>`

   B. `class <generic T> Foo`

   C. `generic class <T> Foo`

   D. `class Foo <T>`

5. ____ **The parameter for the Thread constructor may be**

   A. An object that implements Runnable

   B. Any method

   C. An object that implements Thread

   D. Any method with no parameters

6. ____ **To reliably use class `Bird` as the generic key type for a Java HashMap, `Bird` must**

   A. implement the `Hashable` interface

   B. override the `hashCode()` method

   C. be a primitive type

   D. override the `compareTo()` method

7. ____ **Which of the following demonstrates try-with-resources?**

   A. `try (new MP3Reader(file) != null) {`

   B. `try (new MP3Reader(file)) {`

   C. `try (MP3Reader mp3 = new MP3Reader(file)) {`

   D. `try (new Resource(new MP3Reader(file))) {`

8. ____ **One difference between an Iterator and ListIterator is**

   A. ListIterator (unlike Iterator) can manipulate multiple elements (the "list") one method call

   B. Iterator is just an interface, while ListIterator is the only class that implements Iterator

   C. ListIterator (unlike Iterator) can also iterate backwards

   D. Iterator works with HashMap while ListIterator works with ArrayList

9. ____ **To instance a mutual exclusion object for a `synchronized` scope in Java, write**

   A. `Mutex mutex = new Mutex(new Object());`

   B. `Object mutex = new Object();`

   C. `Thread.mutex mutex = new Thread.mutex(new Object());`

   D. Java uses `synchronized` only with methods, never with mutual exclusion objects

10. ____ **The relationship between overridden `equals(Object o)` and `hashCode()` methods for a class is**

    A. If `equals` is true for two instances they must have the same `hashCode`

    B. The complete default `equals(Object o)` implementation is literally `hashCode() == o.hashCode()`

    C. If two instances have the same `hashCode` then `equals` must be true

    D. These two methods are unrelated and may be independently implemented

11. ____ **To effectively read or write a file, we need to know**

   A. its name and file format
   B. its path, name, and size
   C. its name, access permissions, and inode
   D. its path and name

12. ____ **To pause the current thread until Thread t that it created exits, write**

   A. `t.ends();`
   B. `t.synchronize();`
   C. `t.pauseUntilExit();`
   D. `t.join();`

13. ____ **To ensure that a downcast of the object referenced by superclass variable** `Ball ball` **to subclass variable** `Baseball baseball` **will succeed, use the conditional**

   A. `if(ball isa Baseball) // downcast will succeed!`
   B. `if(ball instanceof Baseball) // downcast will succeed!`
   C. `if(ball canDowncastTo baseball) // downcast will succeed!`
   D. `if(ball.class.equals(Baseball)) // downcast will succeed!`

14. ____ **One difference between a Java HashSet and TreeSet is**

   A. The TreeSet elements are always sorted, HashSet elements are not
   B. The TreeSet works with primitives, the HashSet with objects
   C. The TreeSet is the interface, HashSet implements TreeSet
   D. The TreeSet elements are accessed using two indices rather than one

15. ____ **Subclass Pontiac extends superclass Car. Given** `Pontiac p = new Pontiac();` **which demonstrates successful upcasting?**

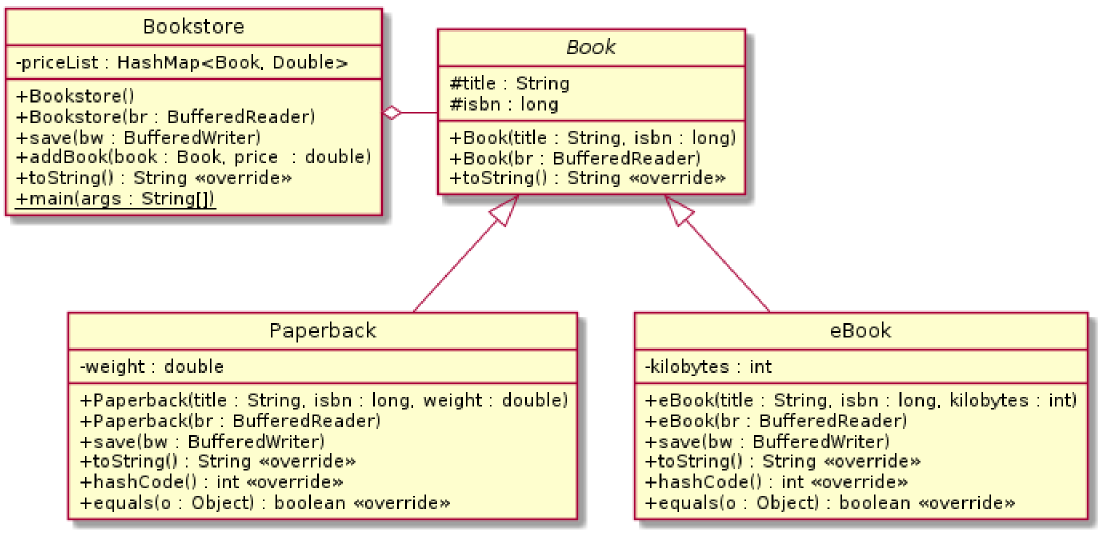
   A. `Pontiac p2 = new Pontiac();`
   B. `Pontiac p2 = (Pontiac) p;`
   C. `Car c = p;`
   D. Upcasting isn't possible with this code

Free Response

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

1. (polymorphism, map, file i/o) A modern bookstore sells books in many formats, including paperback and digital. Consider *and follow* this class diagram, coding only the class members specified below. A Book is defined by its title and ISBN number, with two subclasses - Paperback which includes a shipping weight, and eBook which includes the number of kilobytes it consumes in an electronic reader.

Bookstore manages both in a HashMap, with the Book as key and the price (a Double) as value.


Bookstore
-priceList : HashMap<Book, Double>
+Bookstore()
+Bookstore(br : BufferedReader)
+save(bw : BufferedWriter)
+addBook(book : Book, price : double)
+toString() : String «override»
+main(args : String[])
Book
#title : String
#isbn : long
+Book(title : String, isbn : long)
+Book(br : BufferedReader)
+toString() : String «override»
Paperback
-weight : double
+Paperback(title : String, isbn : long, weight : double)
+Paperback(br : BufferedReader)
+save(bw : BufferedWriter)
+toString() : String «override»
+hashCode() : int «override»
+equals(o : Object) : boolean «override»
eBook
-kilobytes : int
+eBook(title : String, isbn : long, kilobytes : int)
+eBook(br : BufferedReader)
+save(bw : BufferedWriter)
+toString() : String «override»
+hashCode() : int «override»
+equals(o : Object) : boolean «override»

a. {4 points} Write Paperback's hashCode() method. Include all 3 fields when calculating the hash code.

b. {4 points} Write Paperback's equals(Object o) method. Include all 3 fields when comparing objects.

c. {2 point} Write Bookstore's priceList field, which both declares and constructs priceList.

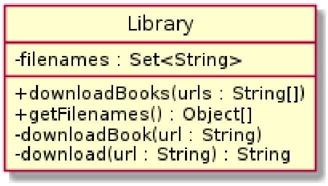
d. {2 point} Write Bookstore's addBook method, which adds a Paperback OR an eBook to the

e. {4 points} **Using an iterator**, write Bookstore's `toString` method, which should return a String that includes both the book (which may be a Paperback or eBook) and price for each entry in field `priceList`, with each entry newline separated, otherwise using any format you like.

f. {5 points} Write the `eBook(BufferedReader br)` constructor, which restores its superclass' and its own fields from the stream.

g. {5 points} Write Bookstore's `main` method, which opens file "test.books" for reading using try-with-resources, constructs a new Bookstore `store` from that stream, and prints `store` to standard output. If an `IOException` occurs, print "Failed to read test.books" to standard error and exit with error code -2.

2. (threads) Class `Library` manages the filenames of PDF books in the local filesystem. Passing an array of url Strings to the `downloadBooks` method causes those books to be downloaded from the Internet, adding their filenames to a Set of Strings (field `filenames`). Method `getFilenames` returns an `Object[]` of the filenames for all downloaded books.


Library
-filenames : Set<String>
+downloadBooks(urls : String[])
+getFilenames() : Object[]
-downloadBook(url : String)
-download(url : String) : String

```
public class Library {
   Set<String> filenames = new HashSet<>(); // Stores filenames of download books

   public void downloadBooks(String[] urls) {
       for(String url : urls) {
           downloadBook(url);
       }
   }
   private void downloadBook(String url) {
       String filename = download(url);
       filenames.add(filename);
   }
   // Other code omitted
}
```

You will modify method `downloadBooks(String[] urls)` to safely call `downloadBook(String url)` for **each** url in a **separate** Java Thread.

a. {3 points} Method `downloadBook` accepts a url and calls method `download` to download the book and add the filename it returns to field `filenames`. **Rewrite** method `downloadBook` to avoid thread interference when adding the returned `filename` to field `filenames`. IMPORTANT: `downloadBook` will be the entire body of each thread, so do NOT simply limit execution of `downloadBook` to one thread at a time!

b. {7 points} Method `downloadBooks` accepts an array of Strings containing the urls for books to be downloaded and passes each one in turn to method `downloadBook`. **Rewrite** this method to download each book using a separate thread. Here are some hints to help you get started:

- You will need an array or collection to reference Thread objects. It will hold exactly one thread per url.
- You will likely need to create a `final` copy of the loop variable or url for starting each Thread.
- You may use a lambda, anonymous class, or separate class as you please to represent the Thread's `run()` method, but as usual a lambda may be much shorter.
- Each Thread, once started, should call `downloadBook` with a single url from the urls parameter. Ensure each url is downloaded exactly once!
- Once all threads have started, ensure they all finish before returning, being careful to handle the InterruptedException checked exception.

3.a. (generics) {4 points} For class Genericize, write the public static generic method print, which accepts a Collection (or subclass of Collection) containing any type, iterates over its values using a for-each or iterator, and prints each value. Once complete, the following code would print "Hello", then "Betty", and finally "Boop" to the console.

```
public class Genericize {

   public static









   }
   public static void main(String[] args) {
       ArrayList<String> strings = new ArrayList<>();
       strings.add("Hello"); strings.add("Betty"); strings.add("Boop");
       print(strings);
   }
}
```

b. (iterators) {5 points} A "square" is an integer multiplied by itself. The first four squares are 1, 4, 9, and 16. Complete the following main method. After the ArrayList definition below, **obtain an iterator** from `squares` and then **use the iterator** to remove all Double values that are NOT in fact square. HINT: For `Double d` the boolean expression `((Math.sqrt(d) % 1) == 0)` will be `true` if d is a square.

```java
public class Squares {
    public static void main(String[] args) {
        ArrayList<Double> squares = new ArrayList<>(Arrays.asList(
             9.0, 3.1415926, 4.0, 2.71828, 3.0, 1.41421, 2.0
       ));

    }
}
```

Bonus

Bonus {+3 points} A Java Set has no `get(int index)` method. In one *brief* sentence or code fragment, explain how to get elements from a Java Set.