Full Name: Utsav Shah

Student ID#: 1002158824

CSE 1325 OBJECT-ORIENTED PROGRAMMING

Exam #2 «---» 16 11 002 «---» Spring 2025

Instructions

1. **You will need your student ID to turn in this exam.**
2. Students are allowed pencils, erasers, and beverage only.
3. All books, bags, backpacks, phones, **smart watches**, and other electronics, etc. must be placed along the walls. **Silence all notifications.**
4. Verify that you have all **10 pages** of the exam.
5. **CLEARLY print your name and student ID** at the top of this page and every additional (pastel) coding sheet you request.
6. **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.
7. If you leave the room, you may not return.
8. 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: Utsav

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 process	A self-contained execution environment including its own memory space
2 reentrant	An algorithm that can be paused while executing, and then safely executed by a different thread
3 garbage collector	A program that runs in managed memory systems to free unreferenced memory
4 reference counter	A managed memory technique that tracks the number of references to allocated memory, so that the memory can be freed when the count reaches zero
5 global	Memory for static fields
6 interface	A reference type containing only method signatures, default methods, static methods, constants, and nested types
7 inheritance	Reuse and extension of fields and method implementations from another class
8 abstraction	Specifying a general interface while hiding implementation details
9 code	Read-only memory for machine instructions
10 synchronized	The ability to control the access of multiple threads to any shared resource

Word List

Abstraction	Algorithm	Class	Class Library	Code
Collection	Concurrency	Encapsulation	Garbage Collector	Generic Programming
Global	Heap	Inheritance	Interface	Iterator
Java Class Library	Mutex	Object	Polymorphism	Process
Reentrant	Reference Counter	Stack	Synchronized	Thread

20

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. A **Which of the following is TRUE about the Java Class Library (JCL)?**

A. A different algorithm is supplied for each collection type, e.g., `Collections.sortArray` and `Collections.sortSet`

B. Java arrays and `String` are defined as part of the JCL

C. All `Set` implementations in the JCL are ordered (that is, sets are always sorted)

[✓ D. The JCL contains generic collections, algorithms, and iterators]

2. C **To delete the last element returned by a Java `ListIterator it` from a collection, write**

A. `removeLast(++it)`

[✓ B. `it.remove()`]

C. `removeLast(it)`

D. `*it = DELETE`

3. D **To protect access to a Java collection such as `ArrayList` from thread interference,**

A. Wrap each access with `synchronized(mutex)`, where `mutex` is any static object

B. Only access the `ArrayList` via a `synchronized` method

C. Use a `Vector` rather than an `ArrayList`

D. Any one of these would work

4. C **Which is TRUE for a thread of execution in a Java program?**

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

B. The maximum number of threads is set by the number of available cores (CPUs)

C. A Thread shares memory with other threads within an operating system process

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

5. C **Which of the following types may NOT be defined as a generic in Java?**

A. class

B. interface

C. enum

D. method

6

6. D To add `element` to a (non-Map) collection such as an `ArrayList x`, write

A. `Collections.add(x, element);`

B. `x.pushBack(element);`

C. `x[x.size()] = element;`

D. `x.add(element);`

7. A Any type may be used to instance a generic in Java EXCEPT

A. Primitive types (because they are not objects)

B. String type (because it's immutable)

C. Generic types (because you can't nest generic types)

D. ANY type (including all of the above) may be used to instance a generic in Java

✗8. A **Given** `Cat cat;`**, the code** `Calico calico = (Calico) cat;` **represents**

A. An upcast

B. An overcast

C. A backcast

[✓ D. A downcast]

9. D **From the statement** `foo.put("pi", 3.14);` **we can infer that** `foo` **was declared as**

A. `ArrayList<String> foo = new ArrayList<Double>();`

B. `Map<Double, String> foo = new Map<>();`

C. `String foo[];`

D. `Map<String, Double> foo = new HashMap<>();`

10. C `if (calico instanceof Cat)` **will**

A. Throw an `InstanceException` if variable `calico` is not of type `Cat`

B. Be true only if `calico` is an instance of class `Cat`

C. Be true if `calico` is an instance of class `Cat` or one of its superclasses

D. Result in a compiler error

8

11. A **To best ensure that a data file is compatible with your code, always check**

A. For an `IncompatibleFileException`

B. The cyclic redundancy check value and the parity bits

[✓ C. The "magic cookie" and the file version number]

D. The file extension and the last modified date from the operating system

12. A **To obtain a Java iterator that can iterate both forward and backwards on** `ArrayList<Integer> al`**, write**

A. `al.listIterator()`

B. `al.reverseIterator()`

C. `al.iterator()`

D. ALL Java iterators can only iterate forward through a collection such as `ArrayList`

13. B **Given superclass** `Cat` **and subclass** `Calico`**, which Java code *may* call method** `meow` **polymorphically?**

[✓ A. `Cat cat = new Calico(); cat.meow();`]

B. `Calico cat = new Cat(); cat.meow();`

C. `Cat cat = new Cat(); cat.meow();`

D. `Calico cat = new Calico(); cat.meow();`

14. A **To be used in a Java try-with-resources, a class must implement** .

A. `TryWithResources`

[✓ B. `Autocloseable`]

C. `Resources`

D. Any of the above

15. D **To handle a checked exception in Java, you may**

A. Use a try-with-resources statement

B. Use a `throws` clause when declaring the method

C. Use a try / catch statement

D. Any one of these would work

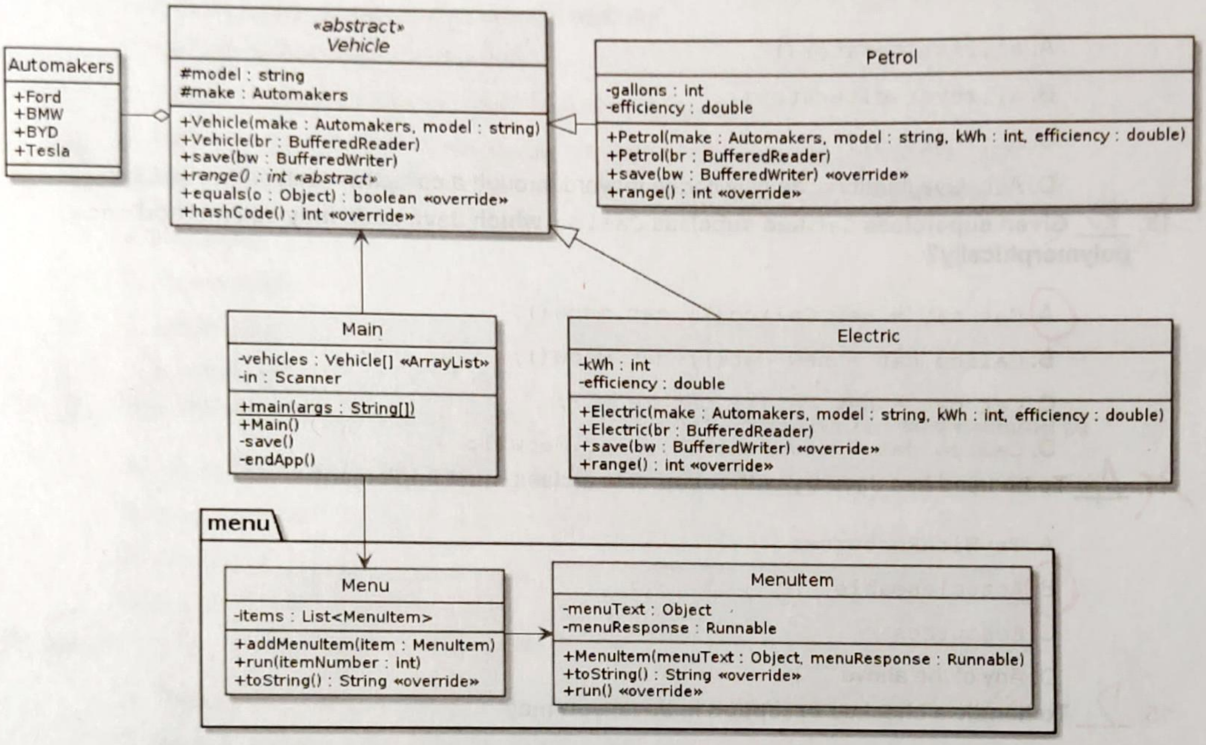
4

Free Response

Provide clear, concise answers to each question. **Write only the code that is requested.** 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.

Additional paper is available on request. Don't write `import` statements unless explicitly asked. *Always* include method declarations when writing method bodies. For methods marked `«override»`, *always* instruct the compiler to verify this.

1. (polymorphism, mdi, file i/o, iterators, equals / hashcode) Consider the following class diagram:


Automakers
+Ford
+BMW
+BYD
+Tesla
«abstract»
Vehicle
#model : string
#make : Automakers
+Vehicle(make : Automakers, model : string)
+Vehicle(br : BufferedReader)
+save(bw : BufferedWriter)
+range() : int «abstract»
+equals(o : Object) : boolean «override»
+hashCode() : int «override»
Petrol
-gallons : int
-efficiency : double
+Petrol(make : Automakers, model : string, kWh : int, efficiency : double)
+Petrol(br : BufferedReader)
+save(bw : BufferedWriter) «override»
+range() : int «override»
Main
-vehicles : Vehicle[] «ArrayList»
-in : Scanner
+main(args : String[])
+Main()
-save()
-endApp()
Electric
-kWh : int
-efficiency : double
+Electric(make : Automakers, model : string, kWh : int, efficiency : double)
+Electric(br : BufferedReader)
+save(bw : BufferedWriter) «override»
+range() : int «override»
menu
Menu
-items : List<MenuItem>
+addMenuItem(item : MenuItem)
+run(itemNumber : int)
+toString() : String «override»
MenuItem
-menuText : Object
-menuResponse : Runnable
+MenuItem(menuText : Object, menuResponse : Runnable)
+toString() : String «override»
+run() «override»

a. {5 points} In file Vehicle.java, write only method `equals`. Only fields `make` and `model` are significant.

```
@Override
public boolean equals (object o) {
    if (o == this) return true;
    if (o == null || o.getClass() != this.getClass()) return false;
    Vehicle v = (Vehicle) o;
    return model.equals(v.model) && make == this.make;
}
```

5

b. {2 points} In file Vehicle.java, write only method `hashCode`. Only fields `make` and `model` are significant.

```
@Override
public int hashCode() {

    return Objects.hash(make, model);

}
```

2

c. {4 points} In file Vehicle.java, write just the `save` method for class `Vehicle` that writes both fields `make` and `model` to the `BufferedWriter` stream parameter. Report that a checked `IOException` may be thrown by `save`. Part of the first line is provided for you.

```
public void save(BufferedWriter bw) throws IOException {
  try(BufferedWriter bw = new BufferedWriter(new FileWriter(filename))) {
      bw.write ("" + make + model + "\n");
  }
}
```

+"\n"

3 1/2

d. {5 points} In file Electric.java, write just the `Electric` constructor that restores the superclass fields and local fields `kWh` and `efficiency` from the `BufferedReader` stream parameter. Report that a checked `IOException` may be thrown by this constructor. Part of the first line is provided for you.

```
public Electric(BufferedReader br) throws IOException {
    super(br);
    this.kWh = Integer.parseInt(br.readLine());
    this.efficiency = Double.parseDouble(br.readLine());
}
```

5

e. {3 points} In file Main.java, import both `Menu` and `MenuItem` from package `menu`. Declare class `Main`, then write the `Main` constructor that instances a `Menu` and add one `MenuItem` instance to it: "Save" which calls method `save()` when `run`. Do NOT write the rest of class `Main` here.

```
package menu.Menu;
Import
package menu.MenuItem;
public class Main {
   public Main() {
        Menu menu = new Menu;
      menu.addMenuItem(new MenuItem("Save", () -> save()));
} }
```

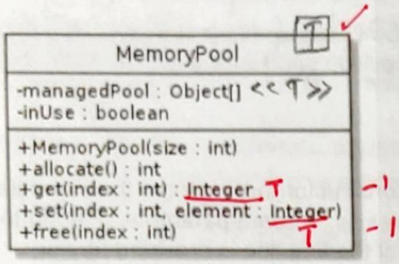
2 1/2

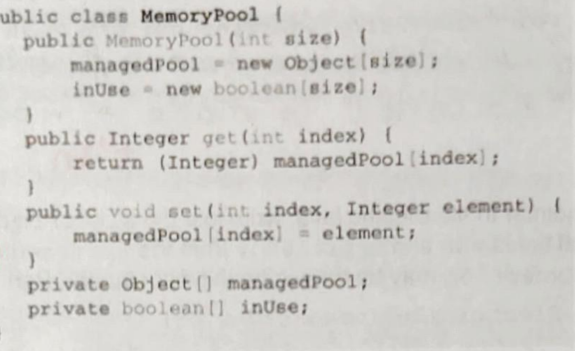
f. {8 points} In file Main.java, write only method `save`. Read a filename from the user using private field `in` (which is already initialized for keyboard input), then open the filename for writing using try-with-resources. **Use an iterator** from field `vehicles` to *polymorphically* call the subclass method `save` for each `Electric` or `Petrol` object in the `vehicles` ArrayList field. If a write error occurs while saving the vehicles, print "Error reading file" to standard error.

```
private void save() {
Scanner in = new Scanner(System.in); in = in.nextLine();
try(BufferedWriter bw = new BufferedWriter(new FileWriter(in))) {
     while(it.hasNext()) {

           vehicle.save(); it.next().save(bw);  -1.5


     }

} catch (Exception e) {
  System.err.println("Error reading file" + e); }
}
```

6.5

2. (generics) Consider the following Java class, which implements a memory pool for `Integer` objects (only a subset of the Java code is shown).


T ✓
MemoryPool
-managedPool : Object[] << T >>
-inUse : boolean
+MemoryPool(size : int)
+allocate() : int
+get(index : int) : Integer T
+set(index : int, element : Integer)
T
+free(index : int)
-1
-1



```
public class MemoryPool {
    public MemoryPool(int size) {
        managedPool = new Object[size];
        inUse = new boolean[size];
    }
    public Integer get(int index) {
        return (Integer) managedPool[index];
    }
    public void set(int index, Integer element) {
        managedPool[index] = element;
    }
    private Object[] managedPool;
    private boolean[] inUse;
}
```

For this question, we will modify this class to manage a memory pool containing any Object-based type.

a. {3 points} Redraw or mark up in place the above class diagram **as a generic class** in the UML. The class will still store `Object` references in array `managedPool`, but will set object `element` of the generic type in method `set` and return the generic type from method `get`.

1

b. {3 points} Rewrite just the declaration for class `MemoryPool` to declare it as a generic class. ✓

3 public class MemoryPool <T> { }

c. {2 points} Rewrite only method `get` to support `MemoryPool` as a generic class. Method `get` should return an object of the generic type from array `managedPool`, which is still of type `Object[]`.

public <T> get (int index) {

2 return <T> managed Pool [index] ;

}

d. {2 points} Rewrite only method `set` to finish making MemoryPool a generic Java class. Method `set` should assign an object of the generic type (the second parameter) to array `managedPool` at the index of the first parameter.

T

public <T> void set(int index, Integer element) {

0 managed Pool [index] = <T> element ;

}

3. (threads) Consider the single-threaded application below.

```
public class FindPrimes {
   public static boolean isPrime(int value) { // true if value is prime number
       if(value < 2) return false;
       for(int i=2; i <= Math.sqrt(value); ++i)
           if(value % i == 0) return false;
       return true;
   }
   private static int first; // smallest int to test if isPrime
   private static int last;  // largest int to test if isPrime

   // a. WRITE static method getInt() to safely provide
   //    ints from first through last to the threads, then -1

   // b. This will be your thread body - REWRITE to check ints
   //    returned your getInt() method above until it returns -1
   private static void search() {
       while(first <= last) {
           if(isPrime(first)) System.out.println(first);
           ++first;
       }
   }

   public static void main(String[] args) {
       first = Integer.parseInt(args[0]); // start at first argument
       last  = Integer.parseInt(args[1]); // stop after last argument

       // c. Create 5 threads each running search()
       search();
       //    Join the threads before exiting
   }
}
```

Rewrite the above program to efficiently use 5 threads to print each prime integer between `args[0]` and `args[1]` exactly once. (Do not copy unmodified code, just add code below as directed below.)

3

a. {4 points} Write static method `getInt()` to return a different `int` between `first` and `last`, inclusive, on each call. **Avoid thread interference**, that is, make `getInt()` thread-safe. Once every int between `first` and `last` has been returned, return -1 on all subsequent calls. ✓

```
      ✓
public static synchronized int getInt(int first, int last) {
                                                   X
  try {
 if  while (first <= last) { -½
      X System.out.println(first);
  return ++first;
     } else return -1;  -½
  } catch (Exception e) {
   System.err.println("Error" + e); }
 Thread.exit(-1);

}
```

$3\frac{1}{2}$

b. {4 points} Rewrite method `search()` to run as the thread body. It should loop, asking for ints from method `getInt()` and printing them if `isPrime` is true, until `getInt()` returns -1. Then exit. `isPrime` and `System.out.println` are both thread-safe. Ensure your `GetInt` is thread-safe, too!

```
public static void Search( ) {
  try { while (getInt() != -1){
          (int var = getInt();)
      if (isPrime(var)) System.out.println(var); }
   catch (Exception e) { System.err.println("Failed" + e); }
   Thread. exit(-1); X -1/2
}
```

c. {5 points} Rewrite static method `main` after parsing `first` and `last` to run 5 threads, each running `search()` (think lambda). Then ensure all threads complete before the program exits. Abort OR print an error message, as you please, on an `InterruptedException`.

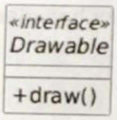
4

```
public static void main(String[] args) {
    try { first = Integer.parseInt(args[0]);
          last = Integer.parseInt(args[1]);
       for(i=0; i<5; i++) { thread = new Thread(() -> thread);
                                                           X
           search ( )); thread.join(); }   start? -1/2
    } catch (InterruptedException e) { System.err.println("Error" + e);}
  Thread. exit(-1);
}
```

-½ put join in separate loop, otherwise only one thred will run at a time!

Bonus

Bonus 1: {+3 points} *Using as little Java code as possible*, write interface `Drawable` shown the class diagram.



$+1\frac{1}{2}$

```
public interface Drawable { }
```

Void draw();

Bonus 2: {+3 points} According to Lecture 10, what was the first user input device for controlling automated computing machines, and in what year was it first used?

0

Paper Tape

enigma was the first user input device for controlling automated computing machines. It was first used in 1955 AD.

WW2 was 1939-1945 ☺