

Full Name: _____

Student ID#: _____

CSE 1325 OBJECT-ORIENTED PROGRAMMING

Exam #2 «---» R 11 001||002||003 «---» 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: _____

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	The provision of a single interface to multiple derived classes, enabling the same method call to invoke different derived methods to generate different results
2	Reuse and extension of fields and method implementations from another class
3	Bundling data and code into a restricted container
4	Specifying a general interface while hiding implementation details
5	A template encapsulating data and code that manipulates it

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

Vocabulary

Word	Definition
1 Polymorphism	The provision of a single interface to multiple derived classes, enabling the same method call to invoke different derived methods to generate different results
2 Inheritance	Reuse and extension of fields and method implementations from another class
3 Encapsulation	Bundling data and code into a restricted container
4 Abstraction	Specifying a general interface while hiding implementation details
5 Class	A template encapsulating data and code that manipulates it

Vocabulary

Word	Definition
6	A reference type containing only method signatures, default methods, static methods, constants, and nested types
7	An instance of a class containing a set of encapsulated data and associated methods
8	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
9	A program that runs in managed memory systems to free unreferenced memory
10	Scratch memory for a thread of execution

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

Vocabulary

Word	Definition
6 Interface	A reference type containing only method signatures, default methods, static methods, constants, and nested types
7 Object	An instance of a class containing a set of encapsulated data and associated methods
8 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
9 Garbage Collector	A program that runs in managed memory systems to free unreferenced memory
10 Stack	Scratch memory for a thread of execution

Vocabulary

Word	Definition
11	Memory shared by all threads of execution for dynamic allocation
12	Memory for static fields
13	Read-only memory for machine instructions
14	Writing algorithms in terms of types that are specified as parameters during instantiation or invocation
15	A library of well-implemented algorithms focused on organizing code and data as Java generics

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

Vocabulary

Word	Definition
11 Heap	Memory shared by all threads of execution for dynamic allocation
12 Global	Memory for static fields
13 Code	Read-only memory for machine instructions
14 Generic Programming	Writing algorithms in terms of types that are specified as parameters during instantiation or invocation
15 Java Class Library	A library of well-implemented algorithms focused on organizing code and data as Java generics

Vocabulary

Word	Definition
16	An object that stores and manages other objects
17	A pointer-like standard library abstraction for objects referring to elements of a container
18	A procedure for solving a specific problem, expressed as an ordered set of actions
19	Performing 2 or more algorithms (as it were) simultaneously
20	A self-contained execution environment including its own memory space

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

Vocabulary

Word	Definition
16 Collection	An object that stores and manages other objects
17 Iterator	A pointer-like standard library abstraction for objects referring to elements of a container
18 Algorithm	A procedure for solving a specific problem, expressed as an ordered set of actions
19 Concurrency	Performing 2 or more algorithms (as it were) simultaneously
20 Process	A self-contained execution environment including its own memory space

Vocabulary

Word	Definition
21	An independent path of execution within a process, running concurrently (as it appears) with other threads within a shared memory space
22	An algorithm that can be paused while executing, and then safely executed by a different thread
23	An object that prevents two properly written threads from concurrently accessing a shared resource
24	The ability to control the access of multiple threads to any shared resource
25	A collection of classes designed to be used together efficiently

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

Vocabulary

Word	Definition
21 Thread	An independent path of execution within a process, running concurrently (as it appears) with other threads within a shared memory space
22 Reentrant	An algorithm that can be paused while executing, and then safely executed by a different thread
23 Mutex	An object that prevents two properly written threads from concurrently accessing a shared resource
24 Synchronized	The ability to control the access of multiple threads to any shared resource
25 Class Library	A collection of classes designed to be used together efficiently

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. ____ 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();`

1. ____ 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();`

CORRECT: A

2. ____ `if (calico instanceof Cat)` **will**

- A. Throw an `InstanceException` if variable `calico` is not of type `Cat`
- B. Be true if `calico` is an instance of class `Cat` or one of its superclasses
- C. Be true only if `calico` is an instance of class `Cat`
- D. Result in a compiler error

2. ____ if (calico instanceof Cat) **will**

- A. Throw an `InstanceOfException` if variable `calico` is not of type `Cat`
- B. Be true if `calico` is an instance of class `Cat` or one of its superclasses
- C. Be true only if `calico` is an instance of class `Cat`
- D. Result in a compiler error

CORRECT: B

(This question was thrown as as most students selected C)

3. ____ **Given** `Cat cat;`, **the code** `Calico calico = (Calico) cat;` **represents**

- A. A downcast
- B. An upcast
- C. A backcast
- D. An overcast

3. ____ **Given** `Cat cat;`, **the code** `Calico calico = (Calico) cat;` **represents**

- A. A downcast
- B. An upcast
- C. A backcast
- D. An overcast

CORRECT: A

4. ____ Which of the following types may NOT be defined as a generic in Java?

- A. method
- B. enum
- C. class
- D. interface

4. ____ Which of the following types may NOT be defined as a generic in Java?

- A. method
- B. enum
- C. class
- D. interface

CORRECT: B

5. ____ **Any type may be used to instance a generic in Java EXCEPT**

- A. Generic types (because you can't nest generic types)
- B. Primitive types (because they are not objects)
- C. String type (because it's immutable)
- D. ANY type (including all of the above) may be used to instance a generic in Java

5. ____ **Any type may be used to instance a generic in Java EXCEPT**

- A. Generic types (because you can't nest generic types)
- B. Primitive types (because they are not objects)
- C. String type (because it's immutable)
- D. ANY type (including all of the above) may be used to instance a generic in Java

CORRECT: B

6. ____ 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. Java *requires* handling a possible `InterruptedException` when joining a Thread
- C. A Thread shares memory with other threads within an operating system process
- D. "Busy loops" like `for(int i=0;i<100000;++i) ;` are a good way to pause a thread
- E. Before using thread-generated data, those threads should be joined to the main thread
- F. Threads speed up algorithms linearly, that is, adding a second thread will cut an algorithm's run time in half
- G. The maximum number of threads is set by the number of available cores (CPUs)
- H. Threads always execute in the order in which they are instanced
- I. A Java thread runs as soon as its Thread object is instanced
- J. A Thread may be paused for at least 2 seconds using `Thread.sleep(2000);`

6. ____ 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. Java *requires* handling a possible `InterruptedException` when joining a Thread
- C. A Thread shares memory with other threads within an operating system process
- D. "Busy loops" like `for(int i=0;i<100000;++i) ;` are a good way to pause a thread
- E. Before using thread-generated data, those threads should be joined to the main thread
- F. Threads speed up algorithms linearly, that is, adding a second thread will cut an algorithm's run time in half
- G. The maximum number of threads is set by the number of available cores (CPUs)
- H. Threads always execute in the order in which they are instanced
- I. A Java thread runs as soon as its Thread object is instanced
- J. A Thread may be paused for at least 2 seconds using `Thread.sleep(2000);`

CORRECT: BCEJ

7. ____ **To handle a checked exception in Java, you may**

- A. Use a `throws` clause when declaring the method
- B. Use a `try / catch` statement
- C. Use a `try-with-resources` statement
- D. Any one of these would work

7. ____ **To handle a checked exception in Java, you may**

- A. Use a `throws` clause when declaring the method
- B. Use a `try / catch` statement
- C. Use a `try-with-resources` statement
- D. Any one of these would work

CORRECT: D

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

- A. Only access the `ArrayList` via a synchronized method
- B. Wrap each access with `synchronized(mutex)`, where `mutex` is any static object
- C. Use a `Vector` rather than an `ArrayList`
- D. Any one of these would work

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

- A. Only access the `ArrayList` via a synchronized method
- B. Wrap each access with `synchronized(mutex)`, where `mutex` is any static object
- C. Use a `Vector` rather than an `ArrayList`
- D. Any one of these would work

CORRECT: D

9. ____ Which of the following is TRUE about the Java Class Library (JCL)?

- A. Java arrays and `String` are defined as part of the JCL
- B. Any non-primitive type may be used as a `Map` key
- C. JCL collections are defined in the `java.collections` module (that is, `import java.collections.ArrayList`)
- D. The number of items to be managed by a collection such as an `ArrayList` must be specified in the constructor, and cannot change thereafter
- E. All `Set` implementations in the JCL are ordered (that is, sets are always sorted)
- F. `ArrayList<Object>` can collect all non-primitive types in a single collection
- G. The JCL contains generic collections, algorithms, and iterators
- H. A different algorithm is supplied for each collection type, e.g., `Collections.sortArray` and `Collections.sortSet`
- I. You may safely subclass (inherit from) JCL classes
- J. The JCL includes both ordered (sorted) and unordered `Map` implementations

9. ____ Which of the following is TRUE about the Java Class Library (JCL)?

- A. Java arrays and `String` are defined as part of the JCL
- B. Any non-primitive type may be used as a `Map` key
- C. JCL collections are defined in the `java.collections` module (that is, `import java.collections.ArrayList`)
- D. The number of items to be managed by a collection such as an `ArrayList` must be specified in the constructor, and cannot change thereafter
- E. All `Set` implementations in the JCL are ordered (that is, sets are always sorted)
- F. `ArrayList<Object>` can collect all non-primitive types in a single collection
- G. The JCL contains generic collections, algorithms, and iterators
- H. A different algorithm is supplied for each collection type, e.g., `Collections.sortArray` and `Collections.sortSet`
- I. You may safely subclass (inherit from) JCL classes
- J. The JCL includes both ordered (sorted) and unordered `Map` implementations

CORRECT: BFGIJ

10. ____ **To add element to a (non-Map) collection such as an ArrayList x, write**

- A. `x.add(element);`
- B. `x[x.size()] = element;`
- C. `x.pushBack(element);`
- D. `Collections.add(x, element);`

10. ____ **To add element to a (non-Map) collection such as an ArrayList x, write**

A. `x.add(element);`

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

C. `x.pushBack(element);`

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

CORRECT: A

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

- A. `Map<Double, String> foo = new Map<>();`
- B. `ArrayList<String> foo = new ArrayList<Double>();`
- C. `String foo[];`
- D. `Map<String, Double> foo = new HashMap<>();`

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

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

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

C. `String foo[];`

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

CORRECT: D

(1 of 4 students were asked this question)

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

A. `al.reverseIterator()`

B. `al.iterator()`

C. `al.listIterator()`

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

(1 of 4 students were asked this question)

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

A. `al.reverseIterator()`

B. `al.iterator()`

C. `al.listIterator()`

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

CORRECT: C

(1 of 4 students were asked this question)

13. ____ **To obtain the previous element from a collection using a Java `ListIterator` `it`, write**

A. `*it--`

B. `*it`

C. `it.previous()`

D. A `ListIterator` cannot obtain the previous element

(1 of 4 students were asked this question)

13. ____ To obtain the previous element from a collection using a Java `ListIterator` `it`, write

A. `*it--`

B. `*it`

C. `it.previous()`

D. A `ListIterator` cannot obtain the previous element

CORRECT: C

(1 of 4 students were asked this question)

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

- A. `*it = DELETE`
- B. `removeLast(it)`
- C. `removeLast(++it)`
- D. `it.remove()`

(1 of 4 students were asked this question)

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

A. `*it = DELETE`

B. `removeLast(it)`

C. `removeLast(++it)`

D. `it.remove()`

CORRECT: D

(1 of 4 students were asked this question)

15. ____ **To insert a new element e before the most recent element returned by a Java**
`ListIterator it`, **write**

A. `it.add(e)`

B. `it -> e`

C. `*it++ = e`

D. A `ListIterator` cannot insert an element before the most recently returned element

(1 of 4 students were asked this question)

15. ____ **To insert a new element e before the most recent element returned by a Java**
`ListIterator it`, **write**

A. `it.add(e)`

B. `it -> e`

C. `*it++ = e`

D. A `ListIterator` cannot insert an element before the most recently returned element

CORRECT: A

16. ____ To overwrite the most recent element returned by a Java `ListIterator` `it` with new element `e`, write

A. `it.set(e)`

B. `it -> e`

C. `*it = e`

D. A `ListIterator` cannot overwrite the most recently returned element

16. ____ To overwrite the most recent element returned by a Java `ListIterator` `it` with new element `e`, write

A. `it.set(e)`

B. `it -> e`

C. `*it = e`

D. A `ListIterator` cannot overwrite the most recently returned element

CORRECT: A

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

- A. The cyclic redundancy check value and the parity bits
- B. The "magic cookie" and the file version number
- C. For an `IncompatibleFileException`
- D. The file extension and the last modified date from the operating system

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

- A. The cyclic redundancy check value and the parity bits
- B. The "magic cookie" and the file version number
- C. For an `IncompatibleFileException`
- D. The file extension and the last modified date from the operating system

CORRECT: B

(This question was thrown out because most students answered D - and D is a reasonable answer!)

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

- A. TryWithResources
- B. Autocloseable
- C. Resources
- D. Any of the above

18. ____ To be used in a Java try-with-resources, a class must implement

A. TryWithResources

B. Autocloseable

C. Resources

D. Any of the above

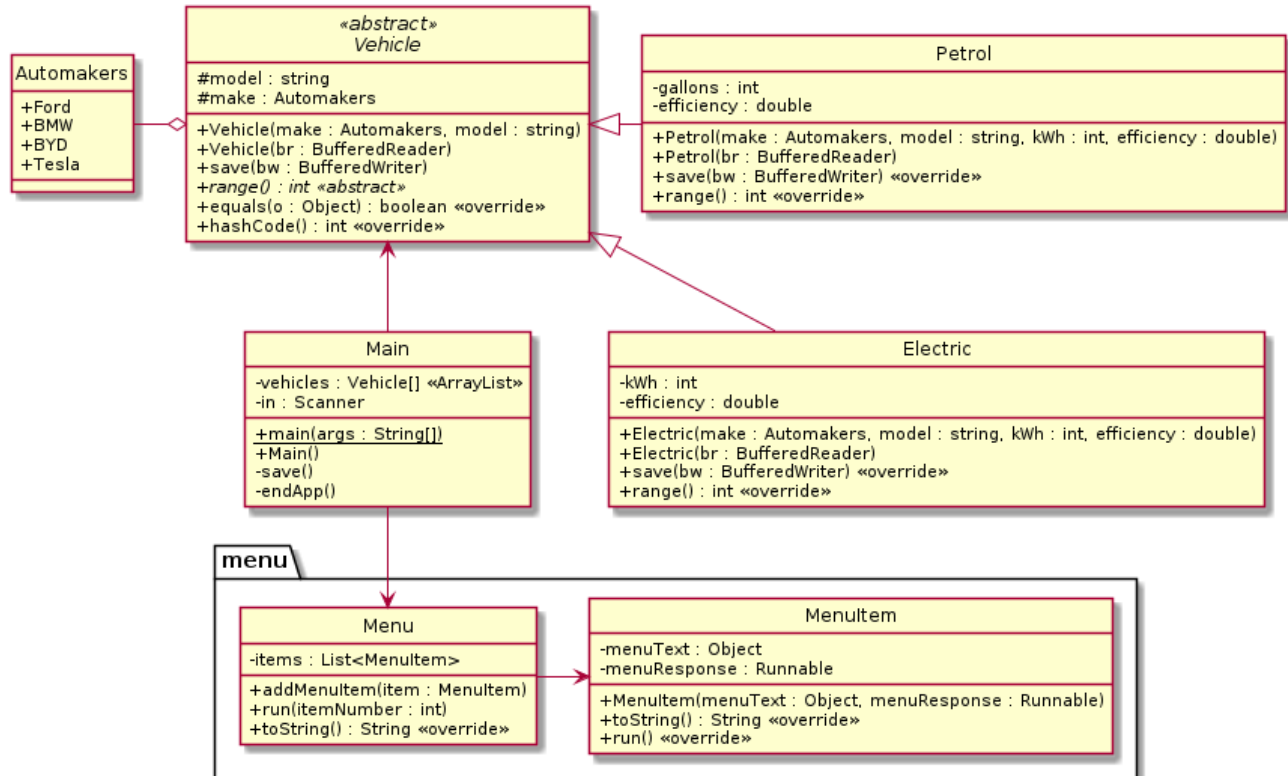
CORRECT: B

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:



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;           // Is it me?
    if(o == null || o.getClass() != this.getClass()) // Is it my type?
        return false;
    Vehicle v = (Vehicle) o;             // Cast!
    return this.make == v.make            // Compare fields
        && this.model == v.model;
}
```

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);
}
```

- 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)
```

```
    public void save(BufferedWriter bw) throws IOException {  
        bw.write("" + make + "\n"); // or bw.write(make.toString() + '\n');  
        bw.write(model + "\n");  
    }
```

- d. {5 points} In file Electric.java || Petrol.java, write just the Electric || Petrol constructor that restores the superclass fields and local fields kWh || gallons 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)
```

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

```
public Petrol(BufferedReader br)
```

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

- 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.

```
import menu.Menu;
import menu.MenuItem;

public class Main {
    public Main() {
        Menu menu = new Menu();
        menu.addMenuItem(new MenuItem("Save", () -> save())); // OR this::save()
    }
}
```

- 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() {
    System.out.print("Filename? "); // OPTIONAL
    try (BufferedWriter bw = new BufferedWriter(new FileWriter(in.nextLine()))) {
        var it = vehicles.iterator();
        // OR Iterator<Vehicle> it = ...
        // OR Iterator it = ...
        while(it.hasNext()) {
            Vehicle v = it.next();
            v.save(bw); // Polymorphism!
        }
        // OR while(it.hasNext()) it.next().save(bw);
    } catch(IOException e) {
        System.err.println("Error reading file");
    }
}
```

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

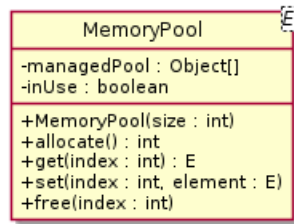
MemoryPool
-managedPool : Object[] -inUse : boolean
+MemoryPool(size : int) +allocate() : int +get(index : int) : Integer +set(index : int, element : Integer) +free(index : int)

```
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;  
}
```

```
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`.



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

```
public class MemoryPool<E> {
```

- 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 E get(int index) {  
    return (E) managedPool[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.

```
public void set(int index, E element) {  
    managedPool[index] = 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.)

- 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.

```
private static synchronized int getInt() {  
    if(first <= last) return first++;  
    return -1;  
}
```

More code but also acceptable:

```
private static Object mutex = new Object(); // static is critical here!  
private static int getInt() {  
    if(first <= last) {  
        synchronized(mutex) {  
            return first++;  
        }  
    }  
    return -1;  
}
```

Since we're not storing these values, we have nowhere to use `Vector` which is inherently synchronized. :(

- 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!

Compact and consistent with the idiom we've been using this semester:

```
private static void search() {  
    int value;  
    while((value = getInt()) >= 0) {  
        if(isPrime(value)) System.out.println(value);  
    }  
}
```

More verbose but also quite acceptable and perhaps easier to understand (most popular on exam):

```
private static void search() {  
    int value = getInt();  
    while(value >= 0) {  
        if(isPrime(value)) {  
            System.out.println(value);  
        }  
        value = getInt();  
    }  
}
```

Only 3 lines of code but harder to understand:

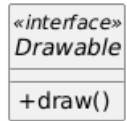
```
private static void search() {  
    for(int i=getInt(); i>0; i=getInt()) {  
        if(isPrime(i)) System.out.println(i);  
    }  
}
```

- 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`.

```
// Create 5 threads here each running search()
Thread[] threads = new Thread[5];
for(int i=0; i< 5; ++i) {
    threads[i] = new Thread(() -> search());
    // threads[i] = new Thread(this::search); does NOT work here!
    threads[i].start();
}
// Join the threads before exiting
for(int i=0; i< 5; ++i)
    threads[i].join();
```


Bonus

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



```
interface Drawable { // NO deduction for "public"
    void draw();      // Deduction for "public" (interface methods default to public)
}
```

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?

"**Paper tape**" or "Papertape" (or "the loop of paper with holes in it" :D)

1725 (+1 if within 10 years, +½ if within the right *century*)

NOTE: If you put **1846** or **1876** exactly as the year, email me and I'll give you credit for that portion of the question given the first two sub-bullets below.

Paper Tape

- Early user interfaces were based on paper tape...
 - Paper tape (with 5, 6, 7, or 24 holes per row) date back to **1725** for automating weaving looms and (in 1876) the "player piano"
 - First used in 1846 to prepare telegrams for transmission
 - Punch machine separate from reader
 - Excellent confetti generator!




Photo by Bad Germ CC-BY-SA-3.0 Unported

Photo by Arnold Reinhold CC-BY-SA-3.0 Unported