```
/**
* This is my code! Its goal is to create an Item object in this abstract class
* CS 312 - Assignment 4
* @author Mari Sisco
* @version 1.0, 10/19/22
*/

abstract class Item
{
    protected String title;
    protected double cost;
    protected boolean isNew;

    /*
     * purpose: initialize a new item
     * input:   the item’s title,cost, and newness flag, isNew
     * result:  the item is initialized
     */
    public Item(String title, double cost, boolean isNew)
    {
        this.title = title;
        this.cost = cost;
        this.isNew = isNew;
    }

    /*
     * purpose: support lookup by cost
     * input:   the target cost, needle
     * result:  true if my cost == needle
     */
    public boolean isMyCost(double needle)
    {
    return cost == needle;
    }

    /*
     * purpose: support lookup by title
     * input:   the target title, needle
     * result:  true if my title == needle
     */
    public boolean isMyTitle(String needle)
    {
        return title.equals(needle);
    }

    /*
     * [ needed for Java’s static typing. non-Books return false ]
     * purpose: determines if a book has a given author
     * input:   the author, needle
     * result:  true if needle is my author
     */

    abstract boolean isMyAuthor(String needle);

    /*
     * [ needed for Java’s static typing, so that item.serialize() compiles. ]
     * purpose: serialize an item
     * input:   only this item
     * result:  the appropriate semi-colon separated string
     */
    abstract String serialize();

    /*
     * purpose: produce a human-readable string representation of the item
     * input:   just the item
     * result:  the item’s string representation
     */
    public String toString()
    {
```

```
        String state = "USED";
    if (isNew)
      state = "NEW";

        return "Title = " + title + "\nCost = $" + cost + "\nCondition = " + state;
    }
}
```

```java
/**
* This is my code! Its goal is to create a Disk object, store its info and imple
ment some useful methods; it extends Item
* CS 312 - Assignment 4
* @author Mari Sisco
* @version 1.0, 10/19/2022
*/

abstract class Disk extends Item
{
    protected int releaseYear;

    /*
     * purpose: initialize a new disk
     * input:   the disk’s title, cost, newness flag (isNew), and year released
     * output:  the initialized Disk
     */
    public Disk(String title, double cost, boolean isNew, int released)
    {
        super(title, cost, isNew);
        releaseYear = released;
    }

    /*
     * purpose: determine if a book has a given auhor
     * input:   the author, needle
     * output:  false, as I am not a book!
     */
    @Override
    public boolean isMyAuthor(String needle)
    {
        return false;
    }

    /*
     * purpose: convert the Disk to a human pleasing string
     * input:   just the disk
     * output:  the disk’s string representation
     */
    @Override
    public String toString()
    {
        return super.toString() + "\nRelease year = " + releaseYear;
    }
}
```

```java
/**
* This is my code! Its goal is to create an Book object, store its info and impl
ement useful methods; It extends Item
* CS 312 - Assignment 4
* @author Mari Sisco
* @version Version 1, 10/19/2022
*/

abstract class Book extends Item
{
    protected String author;

    /*
     * purpose: initialize a new book
     * input:   the book’s title, cost, newness flag (isNew), and author
     * output:  the initialized Book
     */
    public Book(String title, double cost, boolean isNew, String author)
    {
        super(title, cost, isNew);
        this.author = author;
    }

    /*
     * purpose: determine if this book has a given auhor
     * input:   the author, needle
     * output:  true if needle = author
     */
    @Override
    public boolean isMyAuthor(String needle)
    {
        return author.equals(needle);
    }

     /*
     * purpose: convert the Book to a human pleasing string
     * input:   just the book
     * output:  the book’s string representation
     */
    @Override
    public String toString()
    {
        return super.toString() + "\nAuthor = " + author;
    }
}
```

```
/**
* This is my code! Its goal is to create a CD object, store its info and impleme
nt some useful methods; It extends Disk
* CS 312 - Assignment 4
* @author Mari Sisco
* @version Version 1, 10/19/2022
*/

public class CD extends Disk
{
    protected String band;

    /*
     * purpose: initialize a CD
     * input:   the CD's title, cost, newness flag (isNew), year released, and b
and
     * result:  the initialized CD
     */
    public CD(String title, double cost, boolean isNew, int releaseYear, String
band)
    {
        super(title, cost, isNew, releaseYear);
        this.band = band;
    }

    /*
     * purpose: serialize a CD
     * input:   only the CD
     * output:  the appropriate semi-colon representation of the CD
     */
    @Override
    public String serialize()
    {
    String state = "USED";
    if (isNew)
      state = "NEW";
        return title + ";CD;" + cost + ";" + releaseYear + ";" + band + ";" + stat
e;
    }

    /*
     * purpose: generate the string representation of this CD
     * input:   only the CD
     * result:  the string representatio =n of the CD
     */
    @Override
    public String toString()
    {
        return "CD\n" + super.toString() + "\nBand = " + band + "\n";
    }
}
```

```
/**
* This is my code! Its goal is to create a DVD object, store its info and implem
ent some useful methods; It extends Disk
* CS 312 - Assignment 4
* @author Mari Sisco
* @version Version 1, 10/19/2022
*/

public class DVD extends Disk
{
    protected String studio;

    /*
     * purpose: initialize a DVD
     * input:   the DVD's title, cost, newness flag (isNew), year released, and
the studio
     * result:  the initialized DVD
     */
    public DVD(String title, double cost, boolean isNew, int releaseYear, String
 studio)
    {
        super(title, cost, isNew, releaseYear);
        this.studio = studio;
    }

     /*
     * purpose: serialize a DVD
     * input:   only the DVD
     * output:  the appropriate semi-colon representation of the DVD
     */
    @Override
    public String serialize()
    {
        String state = "USED";
        if (isNew)
          state = "NEW";
        return title + ";DVD;" + cost + ";" + releaseYear + ";" + studio + ";" + s
tate;
    }

    /*
     * purpose: generate the string representation of this DVD
     * input:   only the DVD
     * result:  the string representation of the DVD
     */
    @Override
    public String toString()
    {
        return "DVD\n" + super.toString() + "\nStudio = " + studio  + "\n";
    }
}
```

```java
/**
* This is my code! Its goal is to create a PrintBook object, store its info and
implement some useful methods; It extends Book
* CS 312 - Assignment 4
* @author Mari Sisco
* @version Version 1, 10/19/2022
*/
public class PrintBook extends Book
{
    protected String genre;

    /*
     * purpose: initialize a print book
     * input:   the PrinBook's title, cost, newness flag (isNew), author and gen
re
     * result:  the initialized PrintBook
     */
    public PrintBook(String title, double cost, boolean isNew, String author, St
ring genre)
    {
        super(title, cost, isNew, author);
        this.genre = genre;
    }

    /*
     * purpose: serialize a PrintBook
     * input:   only the PrintBook
     * output:  the appropriate semi-colon representation of the book
     */
    @Override
    public String serialize()
    {
        String state = "USED";
        if (isNew)
          state = "NEW";
        return title + ";BOOK;" + cost + ";" + author + ";" + genre + ";" + state
;
    }

    /*
     * purpose: generate the string representation of this book
     * input:   only the book
     * result:  the string representation of the book
     */
    @Override
    public String toString()
    {
        return "BOOK\n" + super.toString() + "\nGenre = " + genre + "\n";
    }
}
```

```java
/**
* This is my code! Its goal is to create an Item object, AudioBooK and store its
 information
* CS 312 - Assignment 4
* @author Mari Sisco
* @version Version 1, 10/19/2022
*/

public class AudioBook extends Book
{
    protected String reader;

    /*
     * purpose: initialize an audio book
     * input:   the AudioBook's title, cost, newness flag (isNew), author and ge
nre
     * result:  the initialized AudioBook
     */
    public AudioBook(String title, double cost, boolean isNew, String author, St
ring reader)
    {
        super(title, cost, isNew, author);
        this.reader = reader;
    }

     /*
     * purpose: serialize an AudioBook
     * input:   only the AudioBook
     * output:  the appropriate semi-colon representation of the AudioBook
     */
    @Override
    public String serialize()
    {
        String state = "USED";
        if (isNew)
          state = "NEW";
        return title + ";AUDIOBOOK;" + cost + ";" + author + ";" + reader + ";" +
 state;
    }

    /*
     * purpose: generate the string representation of this AudioBook
     * input:   only the audio book
     * result:  the string representation of the audio book
     */
    @Override
    public String toString()
    {
        return "AUDIOBOOK\n" + super.toString() + "\nReader = " + reader + "\n";
    }
}
```

```
/**
* This is my code! Its goal is to maintain order on Patrick’s stuff and define m
ethods for some tasks (display, add or remove items)
* CS 312 - Assignment 4
* @author Mari Sisco
* @version Version 1, 10/19/2022
*/

import java.util.Deque;
import java.util.ArrayDeque;

public class Inventory
{
    protected Deque <Item> stuff;

    /*
     * purpose: initialize an empty inventory
     * input:   nothing
     * result:  an empty inventory
     */
    public Inventory()
    {
        stuff = new ArrayDeque<Item>();
    }

    /*
     * purpose: add an item to the inventory
     * input:   the new item, it
     * result:  the inventory is updated
     */
    public void add(Item it)
    {
        stuff.add(it);
    }

    /*
     * purpose: serialize the items of the inventory
     * input:   only the inventory
     * result:  the serialized (semicolon separated) strings with newlines betwe
en items
     */
    public String serialize()
    {
    String ans = "";
    for( Item i : stuff)
          ans +=  i.serialize();
        return ans;
    }

    /*
     * purpose: return the sze of the inventory (used for testing)
     * input:   only the inventory
     * result:  the number of items in the inventory
     */
    public int size()
    {
        return stuff.size();
    }

    /*
     * purpose: display items having a given title
     * input:   the title, needle
     * result:  String reperesentation of matching items
     */
    public String displayMatchingTitle(String needle)
    {
    String display = "";
    for (Item i : stuff)
    {
```

```
      if (i.isMyTitle(needle))
            display += "\n" + i;
    }
        return display;
    }

    /*
     * purpose: display items having a given author
     * input:   the author, needle
     * result:  string representation of the matchin items
     */
    public String displayMatchingAuthor(String needle)
    {
    String display = "";
        for (Item i : stuff)
        {
          if (i.isMyAuthor(needle))
            display += "\n" + i;
        }
        return display;
    }

    /*
     * purpose: display all items
     * input:   nothing
     * result:  string representation of all items
     */
    public String displayAll()
    {
    String display = "";
    if (stuff.size() == 0)
      return display;
    else
    {
          for (Item i : stuff)
          {
            display += "\n" + i.toString();
          }
    }
        return display;
    }

    /*
     * purpose: convert the inventory to a human pleasing string
     * input:   just the inventory
     * result:  the inventory’s string representation
     * [takes use of displayAll() as it does the same thing]
     */
    public String toString()
    {
        return displayAll();
    }

    /*
     * purpose: remove all items with a given author
     * input:   the title, needle
     * result:  the updated inventory
     */
    public void removeMatchingTitle (String needle)
    {
       Deque<Item> copyStuff = new ArrayDeque<Item>(stuff);

       for (Item i : copyStuff)
       {
          if (i.isMyTitle(needle))
            stuff.remove(i);
       }
    }
    /*
     * purpose: remove all items with a given author
     * input:   the author needle
     * result:  the updated inventory
     */
    public void removeMatchingAuthor (String needle)
    {
       Deque<Item> copyStuff = new ArrayDeque<Item>(stuff);

       for (Item i : copyStuff)
       {
          if (i.isMyAuthor(needle))
            stuff.remove(i);
       }
    }
}
```

```
/**
  * This is my code!  Its goal is to create a program to maintain inventory for
  * Patrick's New and Used Stuff Store
  * CS 312 - Assignment 4
  * @author Mari Sisco appending onto Dr.Binkley's code
  * @version 1.0, 10/19/2022
  */

/**
  * This is my code!  Its goal is to provide a command-line interface
  * CS 312 - Assignment 4
  * @author Dave Binkley
  * @version 1.0 10/10/22
  */


public class CLI     // the command line interface!
{
  /*
   * purpose: run the program
   * input:   command from the user (taken from the command line)
   * result:  the database of stuff read from stdin is updated and
   *          written to stdout
   */
  public static void main(String [] args)
  {
    CLI cli = new CLI();
    ItemFactory factory = new ItemFactory();
    Inventory inv = factory.readDatabase(System.in);
    cli.processCommand(args, inv, factory);
  }


  /*
   * purpose: print an error message and the program's command line options
   * input:   an error message
   * result:  message and instructions printed to stdout
   */
  private void usage(String msg)
  {
    System.err.println("\n" + msg + "\nUsage: java CLI [-d|-a|-s] <options>\n"
      + "there are three command line options\n"
      + " (display) -d [(everything by default) | -t title | -a author ]\n"
      + " (add)    -a DVD        \"title\" cost year  \"studio\" NEW|USED\n"
      + " (add)    -a CD         \"title\" cost year  \"band\"   NEW|USED\n"
      + " (add)    -a BOOK       \"title\" cost author  genre    NEW|USED\n"
      + " (add)    -a AUDIOBOOK  \"title\" cost author \"reader\" NEW|USED\n"
      + " (sell)   -s [-t title | -a author]");
  }

  /*
   * purpose: process the user's command
   * input:   the command arguments and the current inventory
   * result:  display requested information or inventory, inv as updated,
   *          is written to stdout
   */
  private void processCommand(String [] args, Inventory inv, ItemFactory factory
)
  {
    if (args.length == 0)
    {
      usage("");
      return;
    }
    if ("-d".equals(args[0]))
    {
      if (args.length == 1)
      {
        System.out.println(inv.displayAll());
      }
      else if ("-t".equals(args[1]) && args.length == 3)
      {
        System.out.println(inv.displayMatchingTitle(args[2].toString()));
      }
      else if ("-a".equals(args[1]) && args.length == 3)
      {
        System.out.println(inv.displayMatchingAuthor(args[2].toString()));
      }
      else
        usage("Invalid display command");
    }
    else if ("-a".equals(args[0]))
    {
      inv.add(factory.createItem(args[1], args[2], Double.parseDouble(args[3]),
args[4], args[5], Boolean.parseBoolean(args[6])));
      System.out.println(inv.serialize());
    }
    else if("-s".equals(args[0]))
    {
      if ("-t".equals(args[1]) && args.length == 3)
      {
        inv.removeMatchingTitle(args[2]);
        System.out.println(inv.displayAll());
      }
      else if ("-a".equals(args[1]) && args.length == 3)
      {
    inv.removeMatchingAuthor(args[2]);
        System.out.println(inv.displayAll());
      }
      else
    usage("Invalid sell command");
    }
    else
    {
      usage("Bummer I don't know how to '" + args[0] + "'");
    }
  }
}
```

```
/**
  * This is my code!  Its goal is to create a program to maintain inventory for
  * Patrick's New and Used Stuff Store
  * CS 312 - Assignment 4
  * @author Mari Sisco appending onto Dr.Binkley's code
  * @version 1.0, 10/19/2022
  */

/**
  * This is my code!  Its goal is to create items
  * CS 312 - Assignment 4
  * @author Dave Binkley
  * @version 1.0 10/10/22
  */


import java.io.BufferedReader;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.io.InputStream;
import java.util.StringTokenizer;

public class ItemFactory     // the maker of Items
{
  public final static int CURRENT_YEAR = 2022;
  public final static int EXPECTED_ARGS = 6;


  // [ an example of the *factory* pattern ]
  /*
   * purpose: create a new item based on the mediaKind
   * input:   the new items data
   * result:  a new Item of the appropriate subclass
   */
  public Item createItem(String title, String mediaKind, Double cost,
                         String authorOrYear, String property2, Boolean isNew)
  {
    Item it = null;
    int year = CURRENT_YEAR;
    int released = -1;

    switch (mediaKind)
    {
      case"AUDIOBOOK":
        it = new AudioBook(title, cost, isNew, authorOrYear, property2);
        break;

      case "DVD":
    released = Integer.parseInt(authorOrYear);
    if (released > year)
    {
          it = new DVD(title, cost, isNew, year, property2);
    }
        else
      it = new DVD(title, cost, isNew, released, property2);
        break;

      case "BOOK":
    if (!property2.equals("SCIFI"))
      property2 = "OTHER";
        it = new PrintBook(title, cost, isNew, authorOrYear, property2);
        break;

      case "CD":
    released = Integer.parseInt(authorOrYear);
        if (released > year)
        {
          it = new CD(title, cost, isNew, year, property2);
        }
        else
          it = new CD(title, cost, isNew, released, property2);
        break;

      default:
        System.err.println("I'll pretend i didn't see the media kind "
                           + mediaKind);
    }

    return it;
  }

  /*
   * purpose: create a new Item based on a database record (line from the file)
   * input:   a semicolon separated string
   * result:  a new Item of the appropriate subclass
   */
  private Item parseItemString(String s)
  {
    StringTokenizer tok = new StringTokenizer(s, ";");
    if (tok.countTokens() != EXPECTED_ARGS)   // [ some defensive programming ]
      return null;  // hey I was promised that the input was valid!
    else
    {
      String [] arr = new String [6];
      for(int i = 0; tok.hasMoreTokens(); i++)
      {
          arr[i] = tok.nextToken();
      }
    return createItem(arr[0], arr[1], Double.parseDouble(arr[2]), arr[3], arr[4]
, Boolean.parseBoolean(arr[5]));
    }
  }

  /*
   * purpose: read the inventory from a Java reader
   * input:   the reader
   * result:  a populated inventory
   */
  public Inventory readDatabase(BufferedReader reader)
  {
    Inventory inv = new Inventory();
    try
    {
      String line;

      for(line = reader.readLine(); line != null; line = reader.readLine())
      {
        if (line.length() == 0)
          continue;   // ignore blank lines

        Item it = parseItemString(line);
        if (it == null)
          System.err.println("Someone needs to take a look at this! " + line);
        else
          inv.add(it);
      }
    }
    catch (Exception E)
    {
      System.err.println("ah sorry but " + E);
    }

    return inv;
  }

  // [ an example of the *wrapper* pattern ]
  /* [ overload the readDatabase method ]
   *
```

```
   * purpose: read the inventory from an input stream
   * input:   the stream, in (e.g., stdin)
   * result:  returns the populated inventory
   */
  public Inventory readDatabase(InputStream in)
  {
    return readDatabase(new BufferedReader(new InputStreamReader(in)));
  }


  // [ another example of the *wrapper* pattern ]
  /* [ overload the readDatabase method ]
   *
   * purpose: read the inventory from a disk file
   * input:   the file name, fileName
   * result:  returns the populated inventory
   */
  public Inventory readDatabase(String fileName)
  {
    try
    {
      return readDatabase(new BufferedReader(new FileReader(fileName)));
    }
    catch (Exception E)
    {
      System.err.println("ah sorry but " + E);
      return null;
    }
  }
}
```