GMU Spring 2023 CS 211 Project 4

Due Date: Sunday, April 23th, 11:59pm

Description

The purpose of this assignment is to practice OOP with exceptions, generics, and collections. Your task is to create a set of classes for managing a **course** catalog. This will require an understanding of exceptions, generics, and the Java collections framework.

Overview

1. Create the Java classes, interfaces, and enumerations described below. Each should be in its own .java file.
2. Implement the methods and fields described.
3. Test your code for correctness.
4. Prepare the assignment for submission and submit it through Gradescope.

Rules

1. This project is an **individual** effort the Honor Code applies.
2. You may import the following classes only: Collection, List, LinkedList, ArrayList, Set, HashSet, EnumSet, Arrays, Collections
3. Comment your code in javadoc style, especially any parts that is not obvious what you are doing.
4. Class fields should not be visible to other classes (unless otherwise noted). Create getter methods if necessary.
5. You may add your own methods, but they must be declared private No additional fields are allowed.

Submission

Submission instructions are as follows:

1. Do not compress your files;
2. Do not use package;
3. Upload all java files in one single submission to Gradescope.

Tasks

You must implement the following classes based on the specification provided. Be reminded that it is very important to follow these specs, otherwise, your code will not compile/run with the tester. If you get errors or warnings from the compiler (e.g., erasure, abstract), do not ignore them.

Course class

This is an abstract class for representing different kinds of courses (Lecture, Laboratory, etc). All the courses in the course catalog will be descendants of this class. It should implement the **Comparable** interface. Specify the generic type for that interface so that **Course** can be compared to another **Course**. However, the details of the **compareTo** method should be left to subclasses.

• **crn**: A string representing the Course Reference Number (CRN) of this course. Should have a getter (final) but no setter.
• **title**: A string representing the title of the course. Should have a getter (final) but no setter. The title length must be at least 15 characters, and no longer than 40 characters.
• **levels**: A collection of strings representing the level of this course. Should have a getter (final) but no setter. Such strings are unique within the collection. At least one level must be provided: Graduate, Non-Degree, and/or Undergraduate.
• **getType()**: A method that will return a string representing the schedule type of this course. The details of this method should be left to subclasses.

• **Course(crn, title, levels)**: constructor with three arguments. Initializes private fields.
• **equals(obj)**: Overrides parent to return true if the given object is also a Course, and their CRNs match. Otherwise, return false.
• **toString()**: returns a string representation of this course. This should have the form "type: *type*, CRN: *crn*, title: *title,* levels: *levels*", replacing the italicized placeholders with the string returned by getType(), crn, title, and levels values.

Collections such as **levels** must be formatted using Arrays.deepToString()

Spacing, spelling, and capitalization are all important. Carefully implementing this method will allow for significant code reuse in subclasses.

MeetDay enumeration

This is an enumeration representing the different Days a course can be dispensed. It should have only the following options:

Monday
Tuesday
Wednesday
Thursday
Friday

LectureCourse class

This is an abstract subclass of Course that represents different types of Lecture courses. It has a **constructor**, **toString**(), and provides an implementation for **compareTo**().

• **instructor**: A string with the name of the instructor. Should have a getter (final) but no setter.
• **credit**: An int indicating the number of credits associated with this course. Should have a getter (final) but no setter.
• **meetDays**: A collection of instances of the MeetDay enum (described above). Should have a getter (final) but no setter. Such instances are unique within the collection. At least one MeetDay must be provided. No more than 2 MeetDays for any courses.
• **gtas**: A collection of strings with the names of the GTAs associated with this course. Should have a getter (final) but no setter. The order of gtas indicates the number of hours they commit to help students in this course. So, the order must be preserved.

• **LectureCourse**(crn, title, levels, instructor, credit, meetDays, gtas): A constructor which sets the private fields . Must call the parent class constructor.

• **toString()**: A string representation of this object. Has the form "instructor: *instructor*, credit: *credit*, meetDays: *meetDays*, gtas: *gtas*," followed by the information given by its parent's toString() method. Consider how inheritance can enable code reuse in this method.

Collections such as **meetDays** and **gtas** must be formatted using Arrays.deepToString().

• **compareTo**(argument): This method returns an integer value representing whether the instance given as an argument should be ordered before or after the calling instance. Negative numbers

indicate the calling instance (this) should be ordered first. Positive numbers indicate the argument should be ordered first. Zero indicates that both instances have the same ordering. This method is what is used to sort collections in the collections framework. See the description of the sort() method in the CourseCatalog class for how this method should be implemented.

InPersonCourse class

This is a concrete class that inherits from **LectureCourse** and represents in-person courses. It overrides the constructor, and getType() methods.

• **getType**(): Returns the string "In-Person”

• **InPersonCourse**(crn, title, levels, instructor, credit, meetDays, gtas):
Calls the parent class constructor with the given arguments.

HybridCourse class

This is a concrete class that inherits from **LectureCourse** and represents hybrid courses. It overrides the constructor, and getType() methods.

• **getType**(): Returns the string "Hybrid”

• **HybridCourse**(crn, title, levels, instructor, credit, meetDays, gtas):
Calls the parent class constructor with the given arguments.

OnlineCourse class

This is a concrete class that inherits from LectureCourse and represents online courses. It overrides the constructor, and getType() methods.

• **getType**(): Returns the string "Online”

• **OnlineCourse**(crn, title, levels, instructor, credit, meetDays, gtas):
Calls the parent class constructor with the given arguments.

AvailableCourse class

This is a generic class, with two type specifications, that represents a mapping between keys and values.

The first type can be any type, the second type must implement the Comparable interface and be comparable to other instances of the same type. The AvailableCourse class itself should also implement Comparable, such that AvailableCourse instances are comparable to other AvailableCourse instances of the same kind (same generic type specifiers). The specific syntax for this class declaration should be:

public class AvailableCourse <K, V extends Comparable<V>> implements Comparable<AvailableCourse <K,V>>

• **key** : A reference of the first generic type. Should have a getter but no setter

• **value** : A reference of the second generic type. Should have a getter but no setter

• **AvailableCourse (key, value):** A constructor which sets the private data members.

• **equals(obj)**: Overrides the parent's method to return true if the other object is also a AvailableCourse and has the same value. Otherwise, returns false

• **compareTo(...)**: Returns the result of comparing the value field. If the generic specification is correct, there should be no need to cast anything.

Searchable interface

This interface contains a single method, matches, which returns true if a given course matches the criteria of this search.

• **matches**(AvailableCourse <String, Course>): this is the method that will be called by the course catalog when searching. Items that return true will be included, but items that return false will not be.

CourseSearcher class

This is a concrete class that implements the Searchable interface, and represents a search with multiple terms, matching courses that contain a given search strings among their fields

• **searchTerms** : a collection of strings to search for in each available course. No getters or setters.

• **CourseSearcher** (searchTerms): constructor that takes a collection of strings and should set searchTerms

• **matches**(AvailableCourse <String, Course>): returns true if the available course contains any of the search terms. Consider using the string returned by the toString method of the Course class. You may find the string method indexOf useful as well.

CourseCatalog class

This class represents the catalog of available courses. It contains a list of courses and provides methods for courses with the CRN it has been registered with, returning a subset of items in the catalog given some search terms, and sorting the catalog.

• **catalog** : A list of available courses. An available course is an instance of the AvailableCourse class with the first type specified to be a string, and the second type specified to be Course AvailableCourse(<String, Course>). Should have a getter (final), no setter.

• **CourseCatalog**(): A default constructor which initializes the catalog

• **add**(crn, course): A method for adding courses to the catalog . The first argument is a string, the second is the course. This method should create a new AvailableCourse<String, Course> instance with the given data.

• **search**(searchable): This method creates and returns a new list of available courses. This new list should contain all available courses from the catalog for which the given searchable's matches method returns true.

• **sort**(): This method sorts ascending the field catalog according to the following rules

1. In-Person courses come before Hybrid courses and Hybrid courses come before Online courses.
2. In-Person courses are sorted first by credit, then by title.
3. Hybrid courses are sorted first by title, then by the number of credits.
4. Online courses are sorted first by title, then by the number of meetDays.

This can be implemented briefly if the compareTo(...) method is correctly implemented in the subclasses of Course and in LectureCourse. The Collections class has a static method sort(...), which will automatically sort using the compareTo(...) method of the elements stored in the list passed as argument. The logic for each compareTo method should first check the type (using instanceof) of the argument, then compare the appropriate strings using their own compareTo (the class String already implements compareTo). As mentioned in the LectureCourse class, a negative value indicates the calling instance should come first, a positive value indicates the calling instance should come second, and zero indicates they have the same ordering.

LectureCourseException class

This is an exception that inherits from Exception and represents situations where:

The constructor of a lecture course receiving arguments that are null, empty collections, or collections with null values.

The exception must implement:

• **fieldName**: A string that stores the name of the null or illegal argument, it must match the name of the field that it was trying to set. Should have a getter but no setter.

• **LectureCourseException(**fieldName): Constructor which sets the private field and calls the parent class constructor with the message "an argument has null or illegal value”

CourseCatalogException class

This is an exception that inherits from IllegalStateException and represents the situation where A course being added to the available course catalog, which CRN is already present in the catalog.

The exception must implement:

• **crn**: A string that stores the CRN of the duplicate course that was being added. Should have a getter but no setter.

• **course** : A reference to the duplicate course. Should have a getter but no setter.

• **CourseCatalogException**(crn, course): Constructor which sets the private fields, and calls the parent class constructor with the message "The course’s CRN is already in the catalog.”.