Core Java Assignment 5

1. Implement a cloneable Person class that has two properties: name and address. The address property should be another object of the Address class that has three properties: street, city, and state. Implement a deep copy mechanism for the Person class, such that when a Person object is cloned, its name and address properties are copied to a new object, and a new Address object is created with the same street, city, and state properties as the original Address object.

2. Create a program that will store and sort a collection of books based on their publication year. Each book has a title, author, and publication year. The program should implement the Comparable interface to provide a natural ordering of books based on their publication year.
    a. Create a Book class that has three properties: title, author, and publicationYear.
    b. Implement the Comparable interface in the Book class so that books can be sorted based on their publication year.
    c. Create a Library class that has a list of Book objects.
    d. Implement a sortBooks() method in the Library class that sorts the list of books based on their publication year using the Collections.sort() method.
    e. Display the sorted list of books to the console.

3. Design a program that sorts an array of Product objects based on their prices using a custom Comparator. The Product class has the following attributes: name (String), price (double), category (String). The program should allow the user to enter the number of products and their attributes, and then sort the array based on their prices in ascending order using a custom Comparator that implements the compare() method.
    a. Define the Product class with the attributes name, price, and category.
    b. Implement the Comparator interface to create a custom comparator that compares Product objects based on their prices.
    c. In the compare() method of the comparator, compare the price attribute of the two Product objects and return a negative integer, zero, or a positive integer depending on whether the first object is less than, equal to, or greater than the second object.
    d. Implement the main method to create an array of Product objects and allow the user to enter their attributes.
    e. Call the Arrays.sort() method to sort the array of Product objects using the custom comparator.
    f. Print the sorted array of Product objects.

4. You have been asked to design a system for a restaurant that can sort a list of menu items based on different criteria, such as name, price, and calorie count. The menu items are represented as objects of a MenuItem class, which contains the following properties:
    a. name (String): The name of the menu item.
    b. price (double): The price of the menu item.
    c. calories (int): The calorie count of the menu item.

   The system should be able to sort a list of MenuItem objects based on any of these properties, using a Comparator. Additionally, the system should be able to sort a list of MenuItem objects based on the natural ordering of the MenuItem class, which is defined by implementing the Comparable interface.

5. Design a class BookCollection that implements the Iterable interface to iterate over a collection of books. The BookCollection class should have the following methods:
    a. addBook(Book book): Add a Book object to the collection.
    b. removeBook(Book book): Remove a Book object from the collection.
    c. getBooks(): Return a list of all the Book objects in the collection.

   In addition, the BookCollection class should implement the Iterator interface to allow iterating over the collection of books. The Iterator should iterate over the books in the order they were added to the collection.

   The Book class should have the following properties:
    a. title: the title of the book
    b. author: the author of the book
    c. publicationYear: the year the book was published

   The Book class should also have a toString() method that returns a string representation of the book in the format: "Title: [title], Author: [author], Year: [publicationYear]".

6. Design a Java program that reads metadata from a given file and displays it to the user. The program should use the Java Reflection API to obtain information about the class and its members, and display the metadata in a user-friendly format.

   The program should take the following steps:
    a. Prompt the user to enter the path of the file containing the metadata.
    b. Use the Java Reflection API to obtain a Class object for the class described in the metadata file.
    c. Display the name of the class to the user.
    d. Display the names and types of the fields and methods defined in the class.
    e. For each field, display its name, type, and any annotations defined on it.

   f. For each method, display its name, return type, parameter types, and any annotations defined on it.

7. Design a Java program that demonstrates the use of Java Reflection to create an object dynamically at runtime, call a method on that object, and access and modify its private fields.

   The program should have a Person class with private fields name and age, and public methods getName, getAge, setName, and setAge. The program should use Java Reflection to create an instance of the Person class, set its private fields, and call its public methods.

8. Design a Java program that uses reflection to inspect and manipulate the fields of a given class. The program should take the following steps:
   a. Ask the user for the name of a class they want to inspect.
   b. Load the specified class using the Class.forName() method.
   c. Use the Class.getDeclaredFields() method to get a list of all fields declared in the class.
   d. Print the name and type of each field to the console.
   e. Ask the user which field they want to manipulate.
   f. Use reflection to set the value of the selected field to a new value entered by the user.
   g. Print the updated value of the selected field to the console.

9. Design a Java program that uses reflection to instantiate a class and invoke its methods dynamically at runtime. The program should have the following features:
   a. The user should be able to enter the fully qualified name of a class, such as "java.util.ArrayList".
   b. The program should use reflection to instantiate the specified class.
   c. The program should display a list of the methods available in the instantiated class.
   d. The user should be able to select a method from the list and provide input parameters if required.
   e. The program should use reflection to invoke the selected method with the provided input parameters.
   f. The program should display the output returned by the method.

10. Design a Java program that uses custom annotations to validate the input parameters of a method. The program should have a custom annotation called @ValidateInput that can be used to annotate method parameters. The @ValidateInput annotation takes two parameters: minValue and maxValue, which define the minimum and maximum allowed values for the parameter.

11. Design a Java program that uses custom annotations to validate the input of a user registration form. The program should include a custom annotation called @ValidRegistration that can be applied to methods that handle user registration requests.
    The @ValidRegistration annotation should validate the input of the registration form according to the following rules:
    a. The username must be at least 6 characters long.
    b. The password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, and one digit.
    c. The email address must be in a valid format.

12. Design a Java program that makes use of custom annotations to validate a data object. The program should have a Person class, which represents a person with the following properties:
    a. id (int): The ID of the person.
    b. name (String): The name of the person.
    c. age (int): The age of the person.
    You should create a custom annotation called ValidatePerson, which has the following constraints:
    a. The id field must be a positive integer.
    b. The name field must not be empty or null.
    c. The age field must be between 0 and 120 (inclusive).
    You should then annotate the Person class with the ValidatePerson annotation, and implement a method that validates the Person object against the constraints specified in the annotation.