CS 5004 Midterm Exam - Spring 2022, Version 3

- This is a **computer-based exam**. That means that you will use your own computers to **individually** work on the problems during the designated exam time, and you will submit your solutions by **first trying to push them to your individual repos on our GitHub course organization. If Github isn't available, please zip/tar your src folder, build.gradle file and any UML diagrams you may want to submit, and upload everytning on Canvas.**
- The exam will be opened between 12 noon PT on Thursday, March 10th until 12 noon PT on Friday, March 11th.
- During your chosen exam day, you are welcome to spend up to **six hours** working on the exam problems.
- When pushing code to your individual repo, you will want to use our **course Gradle project** (the same Gradle project, and the corresponding build file that we have been using throughout this course).
- Additionally, we will use submission rules similar to those we followed for the homework assignments:
  - **Repository content:** Your repositories should contain only the `build.gradle` file and `src` folder with appropriate sub-folders with your code and tests.
  - **Naming convention:** Your solution to each problem should be in its own package. The package names should follow naming convention `midterm.pM`, where you replace M with the problem number, e.g., all your code for problem 1 for this exam must be in a package named `midterm.p1`.
- Some additional expectations and restrictions:
  - **Naming convention:** Please follow the naming convention for classes, methods, variables, constants, interfaces and abstract classes that we have been following throughout this whole course (camel case). Please avoid **magic numbers**.
  - **Immutability:** working in the immutable Java world is not the requirement on this exam.
  - **Methods `hashCode(), equals(), toString()`: if required,** your classes should provide appropriate implementations for methods: `boolean equals(Object o)`, `int hashCode()`, `String toString()`. Appropriate means that it is sufficient to autogenerate these methods, as long as autogenerated methods suffice for your specific implementation. Please don't forget to autogenerate your methods in an appropriate order - starting from the ancestor classes, towards the concrete classes.
  - **Testing your code:** tests are required **only if/where specifically asked for in the assignment**.
  - **Javadoc:** when asked, please include a short description of your class/method, as well as tags `@params` and `@returns` in your Javadoc documentations (code comments). Additionally, if your method throws an exception, please also include a tag `@throws` to indicate that.
  - **UML diagrams:** Please include UML diagrams only when explicitly asked for it.

Good Luck!

Question	Points	Score
1	50	
2	50	
Total:	100	

Problem 1 ***(50 pts)***

(a) You work for an auction company, such as Christie's. Your company is specialized in auctioning art pieces, and you are developing a new listing system that will keep track of all the art pieces your auction company is offering. You are expected to write the part of the system that will automatically calculate starting bids of each art piece based on its characteristics. Another part of the system will eventually use your code to calculate the starting bid at regular intervals. The system calculates the bids for the following types of **art pieces**: ***(10 pts)***

- Paintings
- Photographs
- Non-fungible tokens (NFTs)

Each painting can be one of the following:

- Acrylic painting
- Oil painting
- Water color painting, and
- Charcoal painting.

Each photograph can be one of the following:

- Lifestyle,
- Documentary, and
- Artistic photography.

Each NFT can be one of the following:

- Digital art
- Game
- Music

For each art piece, the system keeps track of:

- Name, represented as a `String`
- Artist/creator names, represented as an array of `Names`, where class name is already provided for you.
- Owners' names, represented as an array of `Names`, where class `Name` is already provided for you.
- Number of days available for auction, encoded as an `Integer`, representing the number of days the art piece has been listed for auction. This will be 0 when it is first listed.
- The latest asking price, represented as `Double`

Additionally, for every painting and every photo, the system also keeps track of:

- Width and height of an art piece, represented as an array of `Doubles`, where the first element represents width, and the second the height.

For every photo, the system also keeps track of the make and model of the camera used to take the photo. Both parameters are encoded as `String`.

For every NFT, the system keeps track of the Ethereum and/or Bitcoin standard used in creating the NFT. This information is encoded as `String`.

Please provide the UML Class diagram for your complete design of the antiques listing system.

(b) The system calculates the starting bid for each art piece according to the following rules: ***(15 pts)***

- Base value of every art piece is equal to its latest asking price, multiplied by a factor of 1.25 (i.e., current value * 1.25).
- The value of a painting and a photograph larger than $12\times$ 16 inches is increased by a factor of 1.2 (i.e., current value * 1.2)).
- The value of a painting auctioned for longer than 100 days, and being offered for more than $2000, decreases by a factor of 0.8 (i.e., current price * 0.8).

- The value of photograph taken with Canon EOS R5, where Canon is a model and EOS R5 is a make of a camera, increases by a factor of 1.35 (i.e., current price * 1.35)).
- The value of meme NFTs, auctioned for more than 15 days decreases by a factor of 0.7 (i.e., current price * 0.7).
- Finally, the value of NFTs, auctioned for more than 60 days, and using Ethereum token standard ERC-721 gets decreased by a factor of 0.6 (i.e., current price * 0.6).

Design and implement method `calculateStartingBid()`, whose return type is `Double`. That means that when we call method `calculateStartingBid()` on some `ArtPiece`, it should return a calculated starting big for that antique as `Double`.

(c) Please provide test class(es) for the class `NFT`, and its subclasses. ***(15 pts)***

(d) The company has grown, and they want to start auctioning two new NFTs: ***(10 pts)***

- Meme
- Video

Please propose your own methods to calculate a starting bid for memes and videos, and provide implementation and a new Class UML Diagram that accommodates these new NFT types.

Problem 2 ***(50 pts)***

(a) You are a part of a team developing a new **national parks' digital directory**. The digital directory of national parks contains information about all of the US national parks. This includes: information about destinations and amenities in the part, lodging and camping information, permits and lotteries resources, as well as relevant information about the current conditions in the park. The proposed directory may in the future be used in an app similar to Recreation.gov. ***(10 pts)***

You team is tasked with specifying the `NationalParkDirectory` ADT, which will be used to store and manage information about the US national parks.

A class `NationalPark`, which stores information about a national park is provided below for your convenience.

Your ADT `NationalParkDirectory` will need to support the following functionality:

- Count the number of `NationalParks` in the `NationalParkDirectory`.
- Check if a specified `NationalPark` is included in the `NationalParkDirectory`.
- Add a new `National Park` to the `NationalParkDirectory`. Please note that `NationalParkDirectory` **does not allow** duplicate `NationalParks`.
- Modify the information about an existing `NationalPark`, for example the information about the national park's area, or about its visitor centers. If the `NationalPark` requested to be modified does not exists in the `NationalParkDirectory`, the system should throw `NationalParkNotFoundException`, which you will have to implement yourself.
- Remove a specified `NationalPark` from the `NationalParkDirectory`. If the `NationalPark` does not exist in the `NationalParkDirectory`, the system should throw `NationalParkNotFoundException`.
- Find and return **any one** `NationalPark` from the `NationalParkDirectory` that is located within the same state, provided as an input argument.
- Find and return **all** `NationalParsk` from the `NationalParkDirectory` that are located within the same state, provided as an input argument.
- Find and return **all** `NationalParks` from the `NationalParkDirectory` that are opened year round.
- Get a `NationalPark` from the `NationalParkDirectorySystem` based upon the `NationalPark`'s unique identifier, `nationalParkID`. The system throws an `InvalidNationalParkIDException` if the given unique ID doesn't exist. You are expected to implement `InvalidNationalParkIDException` for yourselves.

The `NationalParkDirectory` is envisioned as a **data collection** of `NationalParks`, and it can be mutable or immutable. **Please write an interface for the `NationalParkDirectory` ADT. Please include a complete Javadoc for every method.**

(b) The `NationalParkDirectory` is envisioned as a data collection of `NationalParks`. ***(25 pts)***

Please implement `NationalParkDirectory` ADT using an **array-based, sequential or a recursive implementation**. In doing so, please **do not use any built-in data collections from Java Collections Framework, but feel free to use an array, if you wish.** Include a UML class diagram for the complete `NationalParkDirectory` ADT.

(c) Please write white box tests for the following subset of the `NationalParkDirectory` ADT methods: ***(15 pts)***

- Count the number of `NationalParks` in the `NationalParkDirectory`.
- Add a new `National Park` to the `NationalParkDirectory`. Please note that `NationalParkDirectory` **does not allow** duplicate `NationalParks`.
- Find and return all `NationalParks` from the `NationalParkDirectory` that are opened year round.
- Get a `NationalPark` from the `NationalParkDirectory` based upon the `NationalPark`'s unique identifier, `nationalParkID`. The system throws an `InvalidNationalParkIDException` if the given unique ID doesn't exist. You are expected to implement `InvalidNationalParkIDException` for yourselves.

Please note that you **do not have to provide tests** for any other methods of the `NationalParkDirectory` ADT.

```java
package nationalaPark;

import java.time.LocalDate;
import java.util.Arrays;
import java.util.Objects;

/*
Class NationalPark contain information about an individual US national park
 */
public class NationalPark {

  private String nationalParkID;
  private String nationalParkName;
  private String state;
  private Double area;
  private LocalDate dateParkFounded;
  private String[] visitorCenters;
  private Boolean openYearRound;

  public NationalPark(String nationalParkID, String nationalParkName, String state, Double area,
                      LocalDate dateParkFounded, String[] visitorCenters, Boolean openYearRound) {
    this.nationalParkID = nationalParkID;
    this.nationalParkName = nationalParkName;
    this.state = state;
    this.area = area;
    this.dateParkFounded = dateParkFounded;
    this.visitorCenters = visitorCenters;
    this.openYearRound = openYearRound;
  }

  public String getNationalParkID() {
    return nationalParkID;
  }

  public String getNationalParkName() {
    return nationalParkName;
  }

  public String getState() {
    return state;
  }

  public Double getArea() {
    return area;
  }

  public LocalDate getDateParkFounded() {
    return dateParkFounded;
  }

  public String[] getVisitorCenters() {
    return visitorCenters;
  }

  public Boolean getOpenYearRound() {
    return openYearRound;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof NationalPark)) return false;
    NationalPark that = (NationalPark) o;
    return Objects.equals(getNationalParkID(), that.getNationalParkID()) &&
        Objects.equals(getNationalParkName(), that.getNationalParkName()) &&
        Objects.equals(getState(), that.getState()) &&
```

```
            Objects.equals(getArea(), that.getArea()) &&
            Objects.equals(getDateParkFounded(), that.getDateParkFounded()) &&
            Arrays.equals(getVisitorCenters(), that.getVisitorCenters()) &&
            Objects.equals(getOpenYearRound(), that.getOpenYearRound());
    }

    @Override
    public int hashCode() {
      int result = Objects.hash(getNationalParkID(), getNationalParkName(),
          getState(), getArea(), getDateParkFounded(), getOpenYearRound());
      result = 31 * result + Arrays.hashCode(getVisitorCenters());
      return result;
    }

    @Override
    public String toString() {
      return "NationalPark{" +
          "nationalParkID='" + nationalParkID + '\'' +
          ", nationalParkName='" + nationalParkName + '\'' +
          ", state='" + state + '\'' +
          ", area=" + area +
          ", dateParkFounded=" + dateParkFounded +
          ", visitorCenters=" + Arrays.toString(visitorCenters) +
          ", openYearRound=" + openYearRound +
          '}';
    }
}
```