CS 211, ALL SECTIONS
PROJECT 2
DUE SUNDAY, MARCH 1ST AT MIDNIGHT

The objective of this project is to implement a video rental system which will allow people to rent movies and video games. It will require an understanding of classes, inheritence, and overriding methods.

OVERVIEW:

1. Implement the class `Media`, which stores information about rentable media.
2. Derive `Video` and `Game`, both of which are subclasses of `Media`.
3. Implement the class `Payment`, which stores information about credit card payment.
4. Create the `Rental` class, which contains information regarding a particular media rental.
5. Create the `DailyRental` class, a subclass of `Rental` which uses a per-day rental fee policy.
6. Implement the class `RentalBox` which stores and rents out a collection of media.
7. Students in the honors section will have additional functionality to complete for full credit.
8. Download and use the tester module to ensure that your program is correct.
9. Prepare the assignment for submission and submit it.

Many of us have used video rental services to rent DVDs, Blurays, or video games. The implementation of such a system requires an organizational component: the service must keep track of what is available, what is rented, and what to charge people who rent. This is further complicated by the variety of media - video games are not the same thing as movies, and may be organized seperately and have different pricing schemes, yet they all must be maintained by the same system. Here we will see how we can use classes and subclasses to simplify the process.

When we implement this project, we will rely on several Java features. First, we will use classes to organize our data and infuse it with related methods. We will use inheritance to create specialized data structures based on more general ones. Finally, we will use overriding to revise behaviors of some of our methods in subclasses.

RULES

1. This project is an individual effort; the Honor Code applies
2. You will need to import Java's `java.time.*` package in several of your class files (in particular, whereever `LocalDate` is needed). Other imports are off-limits in this project.
3. You may add your own helper methods or data fields, but they should be private or protected. You may add additional constructors which are public.
4. Commenting code is required. You may use JavaDoc-style comments, but for this assignment they are not required to be JavaDoc-style.

MEDIA TASK:
`public class Media`

(*15pts*) The `Media` class represents the media which is rented (including DVDs, Blurays, and video games). This class only represents media in the most general of ways, meaning that it only identifies the media (by name and publication year). More detailed information (such as the type of media) will be saved for subclasses. To serve our purposes, our class will need to implement the following:

- `public Media(String name, int year)` a constructor which initializes the media with the provided name and publication year.
- `public String getName()` retrieves the stored name of the media.
- `public int getYear()` retrieves the stored year of the media.
- `@Override public boolean equals(Object other)` indicates whether this media is the other media which is passed in as a parameter. Much like the `String` class's `equals` method, we define two instances of media as equal if their contents are the same, that is, if their names and publication years match.
  Tip: since the parameter `other` is an `Object` rather than a `Media`, it is recommended to first check whether `other` is an `instanceof Media`, and if it is, cast it to be a `Media` object.
- `@Override public int hashCode()` one of Java's unenforced requirements is that if we ever override an `equals` method, we must also override `hashCode` such that if two object are equal, they also have the same hash code. It is sufficent for this method to return `getName().hashCode()`.
- `@Override public String toString()` displays information about the media as a string, in the format `"NAME (YEAR)"`. For example:

```
The Imitation Game (2014)
```

Video Task:

`public class Video extends Media`

(*10pts*) This class represents a specific type of media: movies. It adds on to the existing `Media` by including information about the movie: the movie's rating (G, PG, R, etc) and runtime (in minutes). Additionally, it contains information about whether the movie is a DVD or a Bluray. This class should provide the following:

- `public static final int DVD = 0;`
- `public static final int BLURAY = 1;`
- `public Video(String name, int year, int runtime, String rating, int format)` a constructor which initializes the movie with the provided name, publication year, runtime in minutes, rating, and format (`DVD` or `BLURAY`).
- `public int getRuntime()` retrieves the stored runtime.
- `public String getRating()` retrieves the stored rating.
- `public int getFormat()` retrieves the stored disc format.
- `@Override public String toString()` displays information about the video as a string, in the format `"`*`NAME`*` (`*`YEAR`*`) `*`FORMAT`*` [`*`RATING`*`, `*`RUNTIME`*` min]"`. Print the format as either `DVD` or `BLURAY`. For example:

```
The Imitation Game (2014) DVD [PG-13, 114 min]
```

Game Task:

`public class Game extends Media`

(*10pts*) Similar to `Video`, we'll also provide a class with information about video games. This class will add onto the basic `Media` information by including information about the platform, rating, number of discs, and whether or not it's an online game. This class requires the following:

- `public Game(String name, int year, String platform, String rating, int discs, boolean online)` a constructor which initializes a game with the given name, publication year, platform, content rating, number of discs, and whether it is an online game.
- `public String getPlatform()` retrieves the stored platform information.
- `public String getRating()` retrieves the stored content rating information.
- `public int getDiscs()` retrieves the stored number of discs information.
- `public boolean isOnline()` retrieves whether or not the game is online.
- `@Override public String toString()` displays information about the game as a string, in the following format: `"`*`NAME`*` (`*`YEAR`*`) `*`PLATFORM`*` [`*`RATING`*`, `*`DISCS`*` discs]"` if it is not an online game, or `"`*`NAME`*` (`*`YEAR`*`) `*`PLATFORM`*` [`*`RATING`*`, `*`DISCS`*` discs, online]"` if it is. For example:

```
Pong (1972) Atari [E, 1 discs]
```

Payment Task:

`public class Payment`

(*10pts*) We will use this class to store a payment method (i.e. credit card information) whenever a person rents something. We need this class to contain appropriate fields plus the following methods:

- `public Payment(String cardNo, String name, int expMonth, int expYear)` initializes the payment method with the provided credit card number, cardholder name, expiration month and expiration year. Behavior for out of bounds dates is not specified - including checks is recommended but will not be tested.
- `public String getCardNo()` returns the stored credit card number.
- `public String getName()` returns the stored cardholder name.
- `public int getExpMonth()` returns the stored expiration month.
- `public int getExpYear()` returns the stored expiration year.
- `@Override public String toString()` returns a string representation of the payment information in the format `"#`*`CARDNUMBER`*` (`*`NAME`*`), exp `*`MONTH`*`/`*`YEAR`*`"`. For example:

```
#0011223344556677 (George Mason), exp 10/2025
```

Rental Task:

`public class Rental`

(*15pts*) The point of having rentable media is to be able to rent it out. Thus, we will create a class which will hold information about a particular rental instance. To rent media, we would need to know what we are renting (the media), the payment method we will use, the date of the rental and the base rental fee. The class will provide us some capabilities related to rentals, such as determining how long the media has been rented, dropping off the rental, determining the total fee, and asking whether the rental has been returned. We assume that the rental period has begun the moment the object is created, and lasts until `dropoff` is called. The current fee can be calculated using `getTotalFee` both before and after the media is dropped off.

The rental is initially rented out, but stops being rented out the moment `dropoff` is called. Thus, our implementation will need to have some way of telling us whether the rental is rented out or returned. There are a number of ways to implement this, including using a `boolean` flag, or using the dropoff date as an indicator.

The class should contain the following:

- `public Rental(Media media, Payment payment, LocalDate today, double fee)` rents the specified media using the provided payment method, with the rental perod beginning on the provided date, using the specified rental fee. When the object is created, the media is assumed to be rented out.
- `public Media getMedia()` retrieves the media which has been rented.
- `public Payment getPayment()` retrieves the payment method used to rent the media.
- `public LocalDate getRentDate()` retrieves the date on which the media was rented.
- `public double getFee()` retrieves the rental fee.
- `public double dropoff(LocalDate today)` drops off the video on the current date and reports the total rental fee (i.e. the fee which we passed in when we created the rental). If the video has not already been dropped off, then this method will set the return date as the current date passed in as a parameter, and then it will return the total fee as given by the `getTotalFee` method. If the video has already been return previously, then this method would have no additional effect, but would still report back the total fee.
- `public boolean isRented()` returns `true` until the first time that `dropoff` is called, after which it returns `false`.
- `public int daysRented(LocalDate today)` if the rental has already been returned, then this method will indicate the total number of days that it was rented. Otherwise, this method will report the total number of days from the rental date until the date provided as a parameter.
  Tip: the following call will allow us to find the span of days between two `LocalDate` objects, `firstDate` and `secondDate`:
  ```
  Period.between(firstDate, secondDate).getDays()
  ```
- `public double getTotalFee(LocalDate today)` computes the total fee for the rental. In this case, it simply returns the flat fee which was passed in when the object was created.
- `@Override public String toString()` returns information about the rental as a string, in the following format: `"`*`MEDIA`*`, rented on `*`DATE`*` using `*`PAYMENT`*`"`
  . For example:
  ```
  The Imitation Game (2014) DVD [PG-13, 114 min], rented on 2019-09-15 using #0011223344556677 (George
  Mason), exp 10/2025
  ```

DailyRental Task:

`public class DailyRental extends Rental`

(*15pts*) A daily rental is a type of rental in which the fees are handled on a daily basis. Instead of having one flat fee, the fee is a daily charge which depends on how many days the video has been rented. Most of this class is the same as the `Rental` class which it derives from, except that the fee calculation is more sophisticated. Thus, it will override the existing `getTotalFee` method. Notice that by doing this, the method will be retroactively replaced whereever it had been used previously, including the `dropoff` method, without any additional action on our part. We will also include the option of providing a credit (i.e. a promo code) which is applied to the rental. This class will contain the following methods:

- `public DailyRental(Media media, Payment payment, LocalDate today, double fee, double credit)` a constructor which will initialize the rental with the given information about media, payment method, rental date, fee (to be interpreted as a daily fee), and promo credit.
- `public DailyRental(Media media, Payment payment, LocalDate today, double fee)` a constructor which will initialize the rental with the given information about media, payment method, rental date, and fee (to be interpreted as a daily fee). The promo credit should default to zero in this version.
- `public double getCredit()` retrieves the promo credit value.
- `@Override public double getTotalFee(LocalDate today)` replaces the flat-rate total fee calculation with a daily fee calculation. The fee will be given by the product of the number of days the video has been rented (*hint: we have a method for that*) and the daily rental fee. Assume that the customer must be charged for at least one day, so if the video has been rented for less than one day, treat it as a 1-day rental. After the daily rental rate is calculated, the promo credit is *subtracted*, with a minimum bound of zero (if subtracting the promo credit results in a negative charge, this method will return zero). Thus, for example, if a video has been rented for 2 days with a fee of `$1.50` and a credit of `$0.50`, then the method will return `2 * $1.50 - $0.50 = $2.50`.

RentalBox Task:

`public class RentalBox`

(*25pts*) Now that we have created the component parts of our rental system, let's design the rental unit itself. This rental box has a fixed capacity and will store and allow us to rent media from it. When we first create it, it will be empty but have a fixed number of slots available to hold media of any type. It will have methods which will allow us to query what is currently available inside of it, and if there are available slots, to stock it with new media. Additionally, the system will be able to check if media is available for rent. If it is, a customer with payment information would be able to rent it out. A new rental would be created and the media would be removed from the system. Finally, if there are slots remaining, a rental can be returned to the system. The class would require the following methods:

- `public RentalBox(int capacity)` creates a rental box with the specified total capacity for `Media` (*hint: use an array*).
- `public RentalBox()` creates a rental box with a default total capacity of `100`.
- `public int boxCapacity()` returns the total media capacity (the number of slots for storing media) contained in this rental unit.
- `public Media get(int i)` returns the media stored in the specified media slot (possibly `null` if the slot is empty). The total number of slots depends on the capacity of the unit.
- `public boolean inStock(Media m)` determines whether the specified media is in stock in the rental unit. Note that it should be sufficient to identify the media by name and year, i.e. using the `equals` method.
- `public boolean put(Media m)` stock the specified media item into the rental unit, if there is space remaining. If there are remaining empty slots in the unit, then the inputted media will be placed in the first available empty slot and the method will return `true`. Otherwise, the method will return `false` and nothing else will happen.
- `public Rental rent(Media m, Payment p, LocalDate d)` creates a new rental based on the choice of media, the payment method, and rental date. If the specified media is not in stock, then this method returns `null`. Otherwise, it will find the first available matching media, remove it from the rental unit, and use it to initialize a new `DailyRental` object. The rental fee should be determined using the `getDailyFee` method. *Important note: once we locate the media in the unit, use that as an argument to the `getDailyFee` call, not the `m` input parameter (because the object from inside the unit may have more info, i.e. whether it is a DVD or a Bluray).*
- `public void processPayment(Payment p, double amount)` prints a message to the standard output indicating how much was charged. The output should be in the format: `"$AMOUNT paid by PAYMENT"`, where the payment amount is printed with two digits after the decimal. For example:

```
$3.00 paid by #0011223344556677 (George Mason), exp 10/2025
```

- `public boolean dropoff(Rental r, LocalDate today)` drops off the current rental. If there are remaining slots in the rental unit and the rental has not already been returned, then this method will place the media in the first available slot, `dropoff` the rental using its own dropoff method, report the total cost of the transaction using the `processPayment` method, before returning `true`. Otherwise, the method will return `false` and take no further action.
- `public double getDailyFee(Media m)` determines the daily fee for the specified media type. For DVDs, this should be `$1.50`, for Blurays it should be `$2.00`, and for video games it should be `$3.00`. Anything else should default to `$0.00`.
  Tip: it will be helpful to use `instanceof` and possibly typecasts to determine the type of media and whether a video is a DVD or a Bluray.
- `@Override public String toString()` displays the contents of the rental unit as a string. This method will display the media in each non-empty slot in the unit, in order of appearance, separated by newlines. Thus, the following is a valid output of this method:

```
The Imitation Game (2014) DVD [PG-13, 114 min]
The Imitation Game (2014) DVD [PG-13, 114 min]
The Imitation Game (2014) DVD [PG-13, 114 min]
The Imitation Game (2014) BLURAY [PG-13, 114 min]
The Imitation Game (2014) BLURAY [PG-13, 114 min]
The Theory of Everything (2014) DVD [PG-13, 124 min]
The Theory of Everything (2014) DVD [PG-13, 124 min]
Pong (1972) Atari [E, 1 discs]
```

Honors Section:

If you are in the honors section, you must complete this part and it is worth 20 points of the project grade. If you are not in the honors section, you are welcome to attempt this but you do not need to complete it and it is not worth any points if you do.

In `RentalBox`, supply two additional methods:

- `public Rental rent(Media m, Payment p, LocalDate d, int format, boolean strict)` similar to the original `rent` method, but it will prefer the specified video format (this method will work but have no special effect if a video game is sought). That is, if the `format` input is `Video.DVD`, it will prefer `DVD` rentals over `BLURAY` rentals if both are available, and vice-versa. If the `strict` flag is `true`, then it will *only* complete the rental if the desired format is available, whereas if it is `false`, it will only prefer the specified format over the other.
- `public String toStringGrouped()` similar to the `toString` method, but the available media is reported grouped by type (first all DVD videos, then all Bluray videos, then all video games). Furthermore, a media item should only be listed at most once per group, so if a particular DVD appears twice, for example, it should only be listed the first time. Other than the grouping and duplicates, all media should still be listed in the order that it appears within the unit.

Testing:

- https://cs.gmu.edu/~iavramo2/classes/cs211/junit-cs211.jar
- https://cs.gmu.edu/~iavramo2/classes/cs211/s20/P2Tester.java

SUBMISSION:

Submission instructions are as follows.

1. Let *xxx* be your lab section number, and let *yyyyyyyy* be your GMU userid. Create the directory `xxx_yyyyyyyy_P2/`
2. Place your files in the directory you've just created.
3. Create the file `ID.txt` in the format shown below, containing your name, userid, G#, lecture section and lab section, and add it to the directory.

   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 004
   Lab section: 213

4. compress the folder and its contents into a .zip file, and upload the file to Blackboard.