CS 211 PROJECT 4: CLASSES AND OBJECTS

DUE MONDAY, APRIL 5TH, 2021, 11:59 PM

- THIS ASSIGNMENT IS WORTH APPROXIMATELY 5% OF YOUR TOTAL GRADE
- PLEASE SUBMIT TO BLACKBOARD

The objective of this project is to implement an application to help manage one's closet. It will heavily test your ability to apply interfaces, and includes elements of exception handling and generics as well.

OVERVIEW:

1. Create the files indicated below and implement the methods described.
2. Download and use a tester module to ensure that your programs are correct. The programs can also be tested using Dr Java's interactive window.
3. Prepare the assignment for submission and submit it.

Files for this Project:

The following files are relevant to the project. Some are provided in the project distribution while others you must create. **You should not submit the files that are provided. DO NOT change any of the data in the provided files.**

File	State	Description
junit-cs211.jar	provided	JUnit library for command line testing
P4Tester.java	provided	Runs all tests
inputData.csv	provided	Data Input File
ID.txt	create	Create in setup to identify yourself
Clothing.java	create	Interface class
Bottoms.java	create	Interface class
Tops.java	create	Abstract class implements Clothing
Pants.java	create	Abstract class implements Clothing, Bottoms
Jeans.java	create	Regular class extends Pants
Legging.java	create	Regular class extends Pants
Shorts.java	create	Regular class extends Pants
JEANFIT.java	create	Enumeration used in Jeans.java
GraphicTshirt.java	create	Regular class extends Tops
Sweater.java	create	Regular class extends Tops
TankTop.java	create	Regular class extends Tops
Graphic.java	create	Regular class used in GraphicTshirts.java
SWEATERTYPE.java	create	Enumeration used in Sweater.java
ClothingException.java	create	Use-Defined Exception Type extends Exception
Label.java	create	Uses Generic Types
InsideCloset.java	create	Regular class that contains Label and Clothing

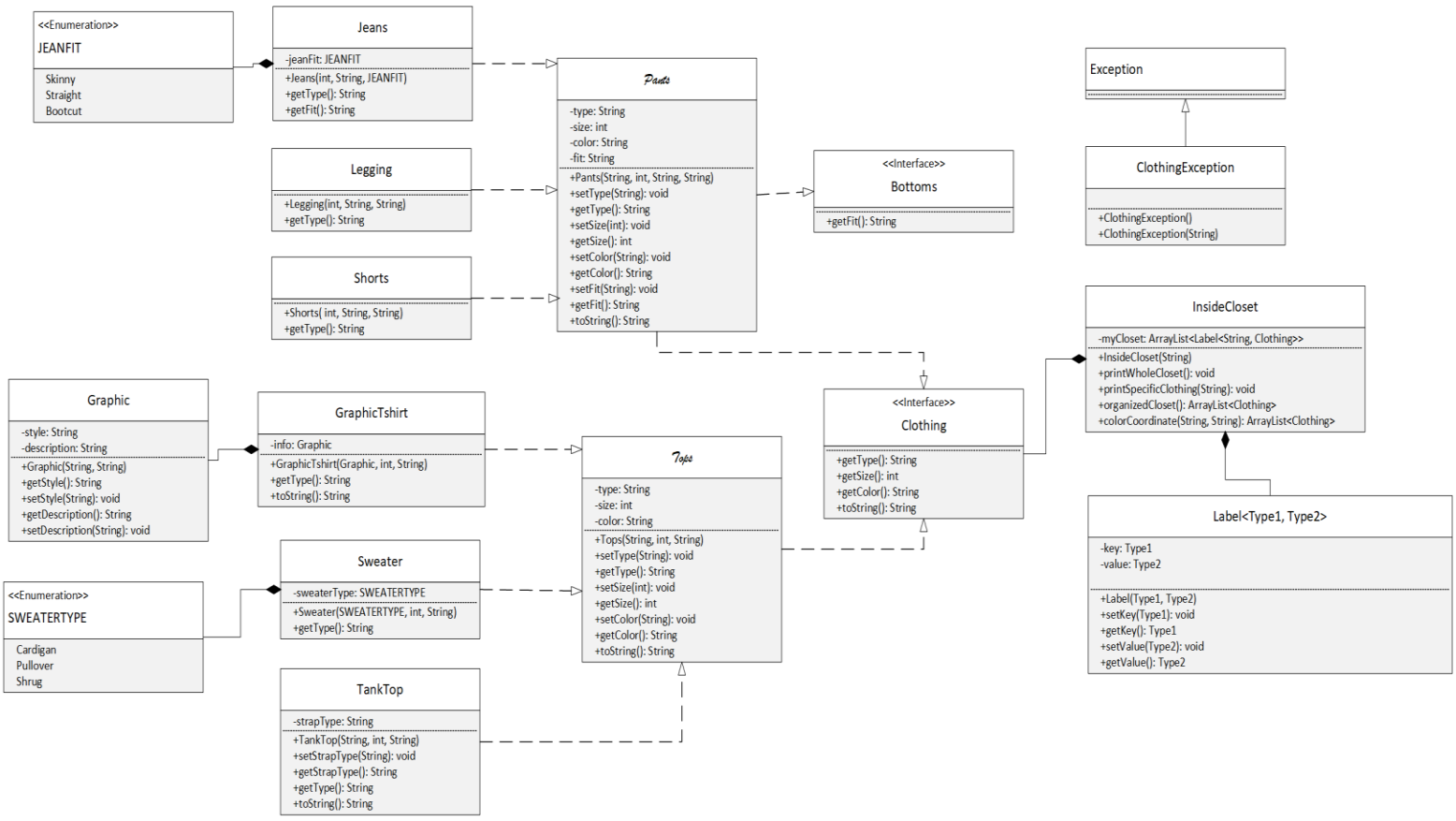
Guidelines and Rules:

- **This project is an individual effort; the Honor Code applies.**
- The main method will not be tested; you may use it any way you want.
- All data fields should be declared private.
- You may add your own helper methods or data fields, but they should be private or protected.
- Format your code clearly and consistently.
- Each class definition will list the fields and methods that must be present. Be sure to match the type and name exactly.
- Use of @Override tags on overridden methods is not required but is strongly recommended.
- When commenting code, use JavaDoc-style comments. Include @param and/or @return tags to explain what is passed in or returned wherever it isn't obvious. Any other JavaDoc tags are optional.

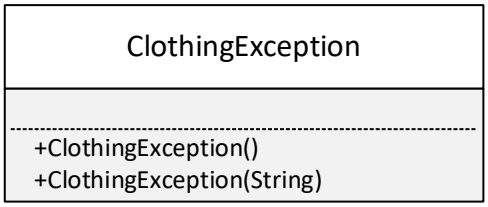
ASSIGNMENT:

While bored at home, you decided to organize your closet. After organizing your closet, you jotted down all the different clothes you have in the closet by type, size, color and other descriptors. You saved this data in a .csv file using Microsoft Excel and saved the file as inputData.csv [provided]. You then got an idea. If you create an application to maintain the data you have gathered, you can use the application to quickly list what is in your closet, without having to physically enter your closet; perhaps you may even list items by type, so you can easily use the information to plan outfits. Based on your idea, you decided to create the below UML diagram.

UML:


<<Enumeration>>
JEANFIT
Skinny
Straight
Bootcut
Jeans
-jeanFit: JEANFIT
+Jeans(int, String, JEANFIT)
+getType(): String
+getFit(): String
Pants
-type: String
-size: int
-color: String
-fit: String
+Pants(String, int, String, String)
+setType(String): void
+getType(): String
+setSize(int): void
+getSize(): int
+setColor(String): void
+getColor(): String
+setFit(String): void
+getFit(): String
+toString(): String
<<Interface>>
Bottoms
+getFit(): String
Exception
ClothingException
+ClothingException()
+ClothingException(String)
Legging
+Legging(int, String, String)
+getType(): String
Shorts
+Shorts(int, String, String)
+getType(): String
InsideCloset
-myCloset: ArrayList<Label<String, Clothing>>
+InsideCloset(String)
+printWholeCloset(): void
+printSpecificClothing(String): void
+organizedCloset(): ArrayList<Clothing>
+colorCoordinate(String, String): ArrayList<Clothing>
Graphic
-style: String
-description: String
+Graphic(String, String)
+getStyle(): String
+setStyle(String): void
+getDescription(): String
+setDescription(String): void
GraphicTshirt
-info: Graphic
+GraphicTshirt(Graphic, int, String)
+getType(): String
+toString(): String
<<Interface>>
Clothing
+getType(): String
+getSize(): int
+getColor(): String
+toString(): String
Tops
-type: String
-size: int
-color: String
+Tops(String, int, String)
+setType(String): void
+getType(): String
+setSize(int): void
+getSize(): int
+setColor(String): void
+getColor(): String
+toString(): String
Label<Type1, Type2>
-key: Type1
-value: Type2
+Label(Type1, Type2)
+setKey(Type1): void
+getKey(): Type1
+setValue(Type2): void
+getValue(): Type2
<<Enumeration>>
SWEATERTYPE
Cardigan
Pullover
Shrug
Sweater
-sweaterType: SWEATERTYPE
+Sweater(SWEATERTYPE, int, String)
+getType(): String
TankTop
-strapType: String
+TankTop(String, int, String)
+setStrapType(String): void
+getStrapType(): String
+getType(): String
+toString(): String

CLOTHINGEXCEPTION



Making an exception class is extremely simple, we would only need to extend our class from the appropriate type of parent exception; and then create constructors which pass on the initialization task to the parent constructors. ClothingException has these requirements:

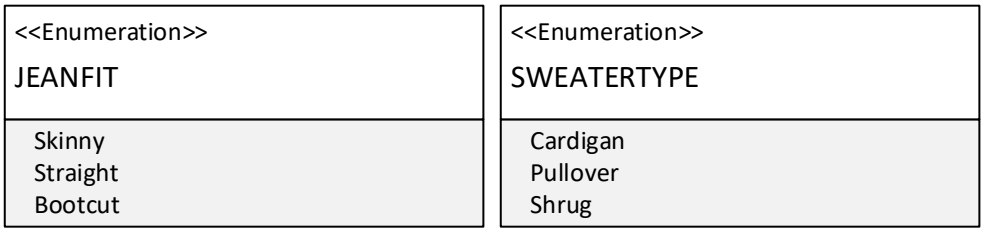
- It extends Exception
- It must have a default constructor
- It must have a constructor which takes a String error message as a parameter. It can simply pass this error message up to the parent constructor.

BOTTOMS, CLOTHING

The below two are interfaces; therefore, they simply list the methods as seen in the UML and do not provide method definitions.

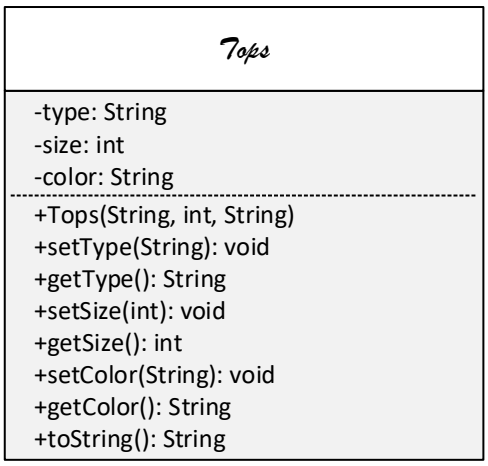
SWEATERTYPE, JEANFIT

The below two are simple Enumerations; therefore, they simply list the values as seen in the UML. Notice the naming of the files JEANFIT.java and SWEATERTYPE.java vs. the enum values Skinny, Straight, Bootcut etc. Pay attention to the uppercase/lowercase.



TOPS

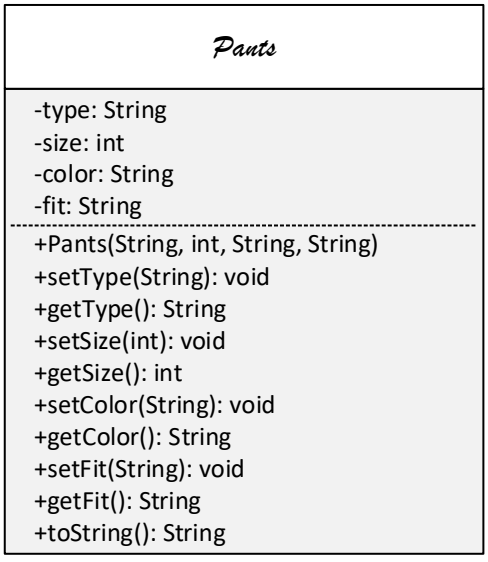
The below class is an abstract class that implements the Clothing interface:



- The abstract Tops class contains three private fields: type, size, color.
- The class contains a constructor used to set the private fields to the passed in values. The parameter order is type, size, color.
- The class contains get/set methods for each private field of the class.
- The Tops class contains a toString() method, which returns the Tops type object in this String format:
  Type: getType() Size: getSize() Color: getColor()
  Ex: Type: Tank Top Size: 13 Color: blue
  Notice how we use the get methods to set up our String rather than directly accessing the fields; this makes it easy for any child classes to use toString().

PANTS

The below class is an abstract class that implements the Clothing interface and the Bottoms interface:

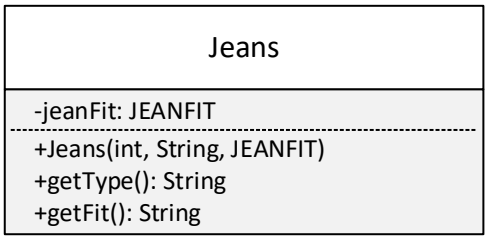


- The abstract Pants class contains four private fields: type, size, color, fit.
- The class contains a constructor used to set the private fields to the passed in values. The parameter order is type, size, color, fit.

- The class contains get/set methods for each private field of the class.
- The Pants class contains a toString() method, which returns the Pants type object in this String format:
  Type: getType() Size: getSize() Color: getColor() Fit: getFit()
  Ex: Type: Shorts Size: 13 Color: blue Fit: Capri
  Notice how we use the get methods to set up our String rather than directly accessing the fields; this makes it easy for any child classes to use toString().

JEANS

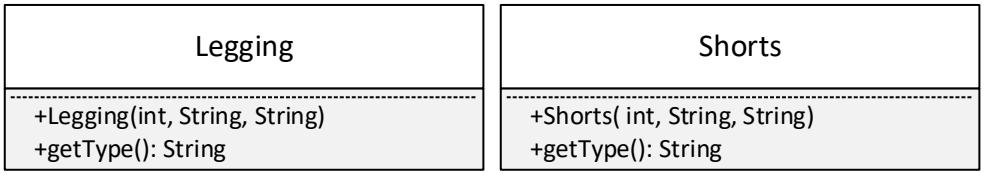
The below class extends Pants:



- The Jeans class adds a new private field: jeanFit of type Enum JEANFIT.
- The Jeans constructor initializes the fields: size, color and jeanFit. Notice type is not set since it is "Jeans" by default.
- getType() returns the literal value "Jeans".
- getFit() returns the String type of the Enum JEANFIT value. i.e returns the String of what the enum is set to: "Skinny" or "Straight" or "Bootcut".

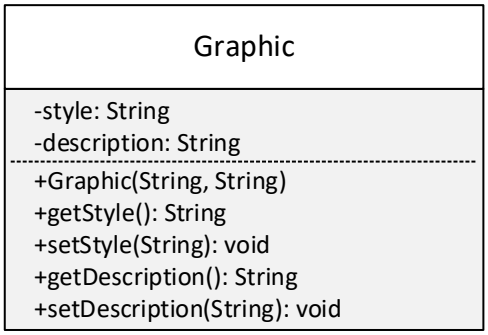
LEGGING, SHORTS

The below two classes both extends Pants:



- The Legging class contains only two methods. A constructor and getType().
- The Shorts class contains only two methods. A constructor and getType().
- The Legging constructor initializes the fields: size, color and fit. Notice type is not set since it is "Legging" by default.
- The Legging getType() method returns the literal value "Legging".
- The Shorts constructor initializes the fields: size, color and fit. Notice type is not set since it is "Shorts" by default.
- The Shorts getType() method returns the literal value "Shorts".

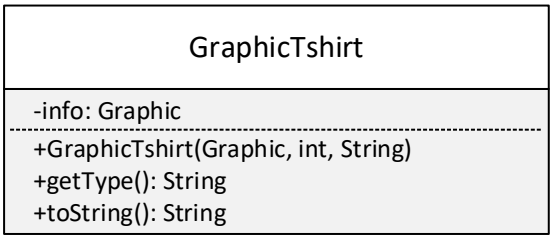
GRAPHIC



- The Graphic class contains two private fields: style, description.
- The class contains a constructor used to set the private fields to the passed in values. The parameter order is style, description.
- The class contains get/set methods for each private field of the class.

GRAPHICTSHIRT

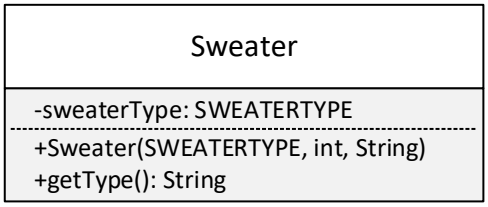
The below class extends Tops:



- The GraphicTshirt class adds a new private field: info of type Graphic.
- The GraphicTshirt constructor initializes the parent class fields: size and color only. Notice type is not set since it is "Graphic T-Shirt" by default. This constructor also initializes the added new private field Graphic info. The parameter order is Graphic info, size, color.
- getType() returns the literal value "Graphic T-Shirt".
- toString() builds on the parent toString() and adds to it the following:
  Graphic Style: info.getStyle() Graphic Description: info.getDescription()
  Ex: Type: Graphic T-Shirt Size: 12 Color: white burgundy Graphic Style: art and design Graphic Description: skull

SWEATER

The below class extends Tops:

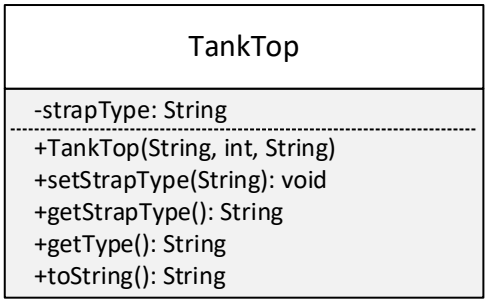


- The Sweater class adds a new private field: sweaterType of Enum SWEATERTYPE.

- The Sweater constructor initializes the parent class fields: size and color only. Notice type is not set since it is one of three types of SWEATERTYPE values. This constructor also initializes the added new private field sweaterType. The parameter order is SWEATERTYPE, size, color.
- getType() returns the String of the Enum SWEATERTYPE value concatenated with the word "Sweater" i.e returns either the String "Sweater(Cardigan)" or "Sweater(Pullover)" or "Sweater(Shrug)".

TANKTOP

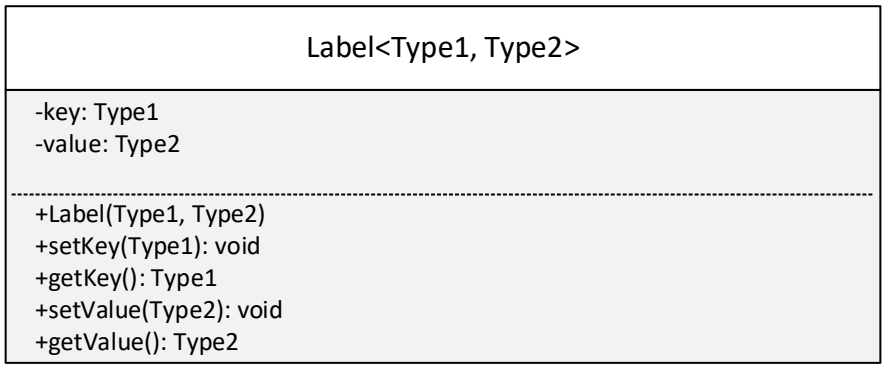
The below class extends Tops:



- The TankTop class adds a new private field: strapType of type String.
- The TankTop constructor initializes the parent class fields: size and color only. Notice type is not set since it is "Tank Top" by default. This constructor also initializes the added new private field strapType. The parameter order is strapType, size, color.
- The TankTop class also contains a get and a set method for the new private attribute strapType.
- getType() returns the literal value "Tank Top".
- toString() builds on the parent toString() and adds to it the following:
  Strap Type: getStrapType()
  Ex: Type: Tank Top Size: 12 Color: white Strap Type: spaghetti

LABEL<TYPE1, TYPE2>

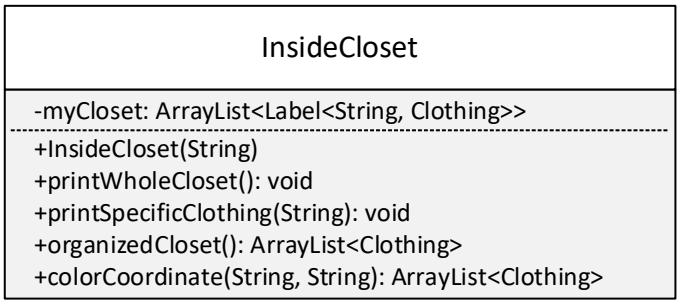
The below class uses generic types:



- The Label class has two generic type private fields: Type1 key, Type2 value.
- The Label constructor initializes the member private fields via passed in generic values.
- The class contains get/set methods for each of the generic private fields respectively.

INSIDECLOSET

InsideCloset is our closet management class that will utilize all of the classes we have created in order to read in data from our .csv file:



- The InsideCloset class has only one private field: myCloset.
  - myCloset is an ArrayList that can contain a list of pairs of values using the Label class; namely a pair of values that are of type String and type Clothing.
- The constructor of this class will read in every data from the passed in String filename. The constructor will read in the .csv file and populate the myCloset ArrayList. For each line read in from the file, the constructor will first determine whether the line of data contains information on a: Graphic T-Shirt, Jeans, Legging, Shorts, Sweater, or Tank Top [this information will always be in the first cell of the row in the .csv file].
- If the row begins with Graphic T-Shirt, the constructor will read in the data following and create a new GraphicTshirt object with the data. The constructor will then add this information to the myCloset ArrayList. The pairs added is: String literal "Graphic T-Shirt" as Label key and the GraphicTshirt object as the Label value.
- If the row begins with Tank Top, the constructor will read in the data following and create a new TankTop object with the data. The constructor will then add this information to the myCloset ArrayList. The pairs added is: String literal "Tank Top" as Label key and the TankTop object as the Label value.
- The same steps as the above is done if the row begin with Jeans, Legging, Shorts, or Sweater (respectively). The string literals used for the Label key are "Jeans", "Legging", "Shorts", "Sweater" respectively. The Label values are Sweater object, Jeans object, Legging object and Shorts object respectively.
- Hint: if you study the .csv file, you will find some interesting information that can help you in separating your data out to store as clothing objects. After reading an entire line of data from the .csv file and using split(",") you can use the following information:
  - For example, for GraphicTShirt: (using array index values)
    - index location 1 always contains the graphic style.
    - index location 2 always contains the graphic description.
    - index location (length-1) always contains the clothing size [the last cell of the row].
    - index locations 3 to length-1 contains all of the colors to be concatenated into a string color field to help create our color attribute for our object.
  - For example, for TankTop: (using array index values)
    - index location 1 always contains the strapType.
    - index locations 2 to length-1 contains all of the colors to be concatenated into a string color field to help create our color attribute for our object.
    - index location (length-1) always contains the clothing size [the last cell of the row].

You can use the above examples to determine how to read in data for Sweater, Jeans, Legging and Shorts as well.

When reading the .csv file, make sure to use try/catch. Don't forget to close the file at the end of the try block. Make sure to throw ClothingException("invalid item") if the value read in from the first cell of a row does not contain a valid value, i.e one of: Graphic T-Shirt, Jeans, Legging, Shorts, Sweater, Tank Top respectively. You should have two catch blocks because of this, one for FileNotFoundException and one for ClothingException.

void printWholeCloset() prints all of the Label values (not Label keys) from the myCloset ArrayList.

This method will use System.out.println to print the data. A sample printout looks like the below:

```
Type: Shorts Size: 13 Color: white black Fit: jegging
Type: Graphic T-Shirt Size: 12 Color: white burgundy Graphic Style: art and design Graphic Description: skull
Type: Tank Top Size: 12 Color: white black Strap Type: wide strap
Type: Graphic T-Shirt Size: 11 Color: black white Graphic Style: music Graphic Description: joan jett
Type: Graphic T-Shirt Size: 10 Color: light blue dark blue Graphic Style: pop culture Graphic Description: little mermaid
Type: Tank Top Size: 13 Color: red blue white Strap Type: spaghetti
Type: Legging Size: 12 Color: gold black Fit: crop
Type: Tank Top Size: 12 Color: white Strap Type: halter
Type: Jeans Size: 11 Color: light blue Fit: Straight
Type: Shorts Size: 11 Color: dark blue Fit: boyfriend
Type: Sweater(Cardigan) Size: 13 Color: blue white
Type: Graphic T-Shirt Size: 13 Color: white purple yellow Graphic Style: pop culture Graphic Description: golden girls
Type: Sweater(Shrug) Size: 13 Color: black orange green blue
Type: Jeans Size: 13 Color: dark blue Fit: Skinny
Type: Sweater(Pullover) Size: 12 Color: yellow blue
Type: Jeans Size: 13 Color: white Fit: Bootcut
Type: Legging Size: 10 Color: black white Fit: full length
Type: Shorts Size: 12 Color: blue Fit: skinny
```

void printSpecificClothing(String) uses System.out.println to print only specific values from the myCloset ArrayList based on the String passed in. If the String passed in is not a valid key (i.e Graphic T-Shirt, Jeans, Legging, Shorts, Sweater, Tank Top), this method throws a ClothingException with the message "invalid item".

Sample printouts look like the below:

A method call printSpecificClothing("Shorts") result in the following:

```
Type: Shorts Size: 12 Color: blue Fit: skinny
Type: Shorts Size: 11 Color: dark blue Fit: boyfriend
Type: Shorts Size: 13 Color: white black Fit: jegging
```

A method call printSpecificClothing("Tank Top") result in the following:

```
Type: Tank Top Size: 13 Color: red blue white Strap Type: spaghetti
Type: Tank Top Size: 12 Color: white black Strap Type: wide strap
Type: Tank Top Size: 12 Color: white Strap Type: halter
```

ArrayList<Clothing> organizedCloset() returns an ArrayList of Clothing objects from the myCloset ArrayList. The returned ArrayList is organized in the order of: Graphic T-Shirt, Jeans, Legging, Shorts, Sweater, Tank Top. If we were to print what this method returns, it will look like the below (compare this to the printWholeCloset() method to see the difference). *Note that this method does not System.out.println the below, it returns an ArrayList of Clothing Objects that we can then use to print the below.*

```
Type: Graphic T-Shirt Size: 12 Color: white burgundy Graphic Style: art and design Graphic Description: skull
Type: Graphic T-Shirt Size: 11 Color: black white Graphic Style: music Graphic Description: joan jett
Type: Graphic T-Shirt Size: 10 Color: light blue dark blue Graphic Style: pop culture Graphic Description: little mermaid
Type: Graphic T-Shirt Size: 13 Color: white purple yellow Graphic Style: pop culture Graphic Description: golden girls
Type: Jeans Size: 11 Color: light blue Fit: Straight
Type: Jeans Size: 13 Color: dark blue Fit: Skinny
Type: Jeans Size: 13 Color: white Fit: Bootcut
Type: Legging Size: 12 Color: gold black Fit: crop
Type: Legging Size: 10 Color: black white Fit: full length
Type: Shorts Size: 13 Color: white black Fit: jegging
Type: Shorts Size: 11 Color: dark blue Fit: boyfriend
Type: Shorts Size: 12 Color: blue Fit: skinny
Type: Sweater(Cardigan) Size: 13 Color: blue white
Type: Sweater(Shrug) Size: 13 Color: black orange green blue
Type: Sweater(Pullover) Size: 12 Color: yellow blue
Type: Tank Top Size: 12 Color: white black Strap Type: wide strap
Type: Tank Top Size: 13 Color: red blue white Strap Type: spaghetti
Type: Tank Top Size: 12 Color: white Strap Type: halter
```

ArrayList<Clothing> colorCoordinate(String item, String color) returns an ArrayList of Clothing objects from the myCloset ArrayList that contain the color of the item passed in. Let's assume we wanted to buy a new Graphic T-Shirt, Jeans, Legging, Shorts, Sweater or a Tank Top; we will pass into this method one of the six items we want to purchase and the color of the item (only a single-color value). This method will take that information and return all the Clothing objects in our myCloset ArrayList that contain that color. However, when the new item we want to purchase is an instanceof Tops, this method returns all the Pants objects that have the same color (this helps us coordinate our oufits!). When the new item we want to purchase is an instanceof Pants, this method returns all the Tops objects that contain the color passed in. If the item passed in is not a valid value, this method throws a ClothingException with the message "invalid item". If nothing in the closet contains the color passed in and our Clothing ArrayList to return is empty, this method throws a ClothingException with the message "nothing of that color in your closet".

If we were to print what this method returns, it will look like the below:
Note that this method does not System.out.println the below, it returns an ArrayList of Clothing Objects that we can then use to print the below.

Method call to colorCoordinate("Tank", "blue") result in:
```
ClothingException: invalid item
```
Method call to colorCoordinate("Tank Top", "teal") result in:
```
ClothingException: nothing of that color in your closet
```
Method call to colorCoordinate("Tank Top", "white") result in:
```
Type: Shorts Size: 13 Color: white black Fit: jegging
Type: Jeans Size: 13 Color: white Fit: Bootcut
Type: Legging Size: 10 Color: black white Fit: full length
```

Method call to colorCoordinate("Shorts", "burgundy") result in:
```
Type: Graphic T-Shirt Size: 12 Color: white burgundy Graphic Style: art and design Graphic Description: skull
```

GRADING:

Rubric:

- Bottoms Class, Clothing class (4 pts)
- Pants, Tops classes (10 pts)
- Jeans, Legging, Shorts classes (15 pts)
- GraphicTshirt, Sweater, TankTop classes (15 pts)
- JEANFIT, SWEATERTYPE (2 pts)
- Graphic class (2 pts)
- ClothingException class (2 pts)
- Label class (10 pts)
- InsideCloset class (30 pts)
- Documentation and Readability (5 pts) [JavaDoc style Commenting, Proper Indentation/Overall Structure]
- Correct Project Setup (5 pts) [zip format, file name, directory structure, files included, ID.txt]

To receive credit for this project you must submit it to Blackboard. Credit will be assigned based on the fraction of tests you pass. You must test your class with the provided tester to ensure that your code can compile and execute. Keep in mind, though, that the grader will have more tests than the provided tester and most of them will be different from the tester. Moreover, **it is your responsibility to verify that the tests pass on the command line, not on IDEs.** Test failures during grading will receive no credit and will not be examined for regrading.

Failure to submit your work in an appropriately named directory and with the ID.txt file in it with correct information will result in a 5% grade deduction.

After verifying that your code runs, zip your directory, and submit it to Blackboard. You may submit as many times as you like; only the final valid submission will be graded. If you turn in late work within 48 hours, there is a penalty (see syllabus). If you turn in late work beyond 48 hours, the submission will be ignored, and you will get zero points.

You should **always verify that your submission is successful**. Go back to the assignment and download your last submission; is there actually a file there? Can you unzip that file and see .java files in there as you would expect? Can you run those files? It turns out you can do quite a bit to ensure you have properly submitted your work. Do not get a zero just because of a failure to turn in the right files! It is your responsibility to correctly turn in your work.

TESTING:

Download the following files provided in the Project 4 assignment link in Blackboard:

- junit-cs211.jar
- P4Tester.java

This is a unit tester which is what we will use to test to see if your classes are working correctly.

The tester file may not be provided on the same day as the assignment instruction is posted. An announcement is made when the tester file is available.

When you think your classes are ready, do the following from the command prompt to run the tests.

On Windows:

```
javac -cp .;junit-cs211.jar *.java
java -cp .;junit-cs211.jar P4Tester
```

On Mac or Linux:

```
javac -cp .:junit-cs211.jar *.java
java -cp .:junit-cs211.jar P4Tester
```

In Dr Java:

- open the files, including `P4Tester.java`, and click the *Test* button.

In Eclipse:

- create a new JUnitTest Case (File --> New --> JUnit TestCase)
- Name it P4Tester
- Copy/paste the content of P4Tester.java in the new file
- Save the file
- Right click on the P4Tester.java on the Package explorer
- Chose Run As -->JUnit Test

SUBMISSION:

Submission instructions are as follows.

1. Let *xxx* be your lab section number and let *yyyyyyyy* be your GMU userid (netid). Create the directory `xxx_yyyyyyyy_P4/`
2. Place all of the `.java` files that you've created into the directory.
3. Create the file `ID.txt` in the format shown below, containing your name, netid, G#, lecture section and lab section, and add it to the directory.
   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 001
   Lab section: 213

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